Package ‘copula’

June 18, 2017

Version 0.999-17

VersionNote Last CRAN: 0.999-16 on 2017-01-07

Date 2017-06-17

Title Multivariate Dependence with Copulas

Author Marius Hofert <marius.hofert@uwaterloo.ca>,
Ivan Kojadinovic <ivan.kojadinovic@univ-pau.fr>,
Martin Maechler <maechler@stat.math.ethz.ch>, and
Jun Yan <jun.yan@uconn.edu>

Maintainer Martin Maechler <maechler@stat.math.ethz.ch>

Depends R (>= 3.1.0)

Imports stats, graphics, methods, stats4, Matrix, lattice, colorspace, gsl, ADGofTest, stabledist (>= 0.6-4), mvtnorm, pcaPP, pspline, numDeriv

Suggests MASS, KernSmooth, sfsmisc, scatterplot3d, Rmpfr, bbmle, knitr, parallel, gridExtra, lcopula, mvnormtest, partitions, polynom, qrng, randtoolbox, rugarch, Runuran, tseries, VGAM, VineCopula, zoo

SuggestsNote the last lines' packages {parallel, ..., zoo} are only used in vignettes, demos and few tests.

Enhances nor1mix

Description Classes (S4) of commonly used elliptical, Archimedean, extreme-value and other copula families, as well as their rotations, mixtures and asymmetrizations. Nested Archimedean copulas, related tools and special functions. Methods for density, distribution, random number generation, bivariate dependence measures, Rosenblatt transform, Kendall distribution function, perspective and contour plots. Fitting of copula models with potentially partly fixed parameters, including standard errors. Serial independence tests, copula specification tests (independence, exchangeability, radial symmetry, extreme-value dependence, goodness-of-fit) and model selection based on cross-validation. Empirical copula, smoothed versions, and non-parametric estimators of the Pickands dependence function.

License GPL (>= 3) | file LICENCE

ByteCompile yes

VignetteBuilder knitr

Collate AllClass.R Classes.R AllGeneric.R Auxiliaries.R aux-acopula.R asymCopula.R mixCopula.R rotCopula.R Copula.R special-func.R amhCopula.R claytonCopula.R frankCopula.R cop_objects.R nacopula.R dC-dc.R amhExpr.R An.R archmCopula.R cCopula.R claytonExpr.R ellipCopula.R empcop.R empPsi.R acR.R estimation.R evCopula.R evTests.R exchTests.R fgmCopula.R fitCopula.R fitLambda.R fitMvdc.R fixedPar.R frankExpr.R galambosCopula.R galambosExpr-math.R galambosExpr.R ggraph-tools.R pairsRosenblatt.R prob.R gofTrafos.R gofEVTests.R gofCopula.R graphics.R gumbelCopula.R gumbelExpr.R huslerReissCopula.R huslerReissExpr.R indepCopula.R indepTests.R joeCopula.R K.R logseries.R mvdc.R matrix_tools.R normalCopula.R opower.R plackettCopula.R plackettExpr.R rstable1.R safeUroot.R schlatherCopula.R stable.R timing.R tCopula.R tawnCopula.R tawnExpr.R tevCopula.R varianceReduction.R wrapper.R xvCopula.R zzz.R

Encoding UTF-8

URL http://copula.r-forge.r-project.org/

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-06-18 13:21:09 UTC

R topics documented:

`copula-package` *Multivariate Dependence Modeling with Copulas*

Description

The **copula** package provides (S4) classes of commonly used elliptical, (nested) Archimedean, extreme value and other copula families; methods for density, distribution, random number generation, and plots.

Fitting copula models and goodness-of-fit tests. Independence and serial (univariate and multivariate) independence tests, and other copula related tests.

Details

The DESCRIPTION file:

Package:	copula
Version:	0.999-17
VersionNote:	Last CRAN: 0.999-16 on 2017-01-07
Date:	2017-06-17
Title:	Multivariate Dependence with Copulas
Author:	Marius Hofert <marius.hofert@uwaterloo.ca>, Ivan Kojadinovic <ivan.kojadinovic@univ-pau.fr>, Martin
Maintainer:	Martin Maechler <maechler@stat.math.ethz.ch>
Depends:	R (>= 3.1.0)
Imports:	stats, graphics, methods, stats4, Matrix, lattice, colorspace, gsl, ADGofTest, stabledist (>= 0.6-4), mvtnorm
Suggests:	MASS, KernSmooth, sfsmisc, scatterplot3d, Rmpfr, bbmle, knitr, parallel, gridExtra, lcopula, mvnormtest,
SuggestsNote:	the last lines' packages parallel, ..., zoo are only used in vignettes, demos and few tests.
Enhances:	nor1mix
Description:	Classes (S4) of commonly used elliptical, Archimedean, extreme-value and other copula families, as well a
License:	GPL (>= 3) | file LICENCE
ByteCompile:	yes
VignetteBuilder:	knitr
Collate:	AllClass.R Classes.R AllGeneric.R Auxiliaries.R aux-acopula.R asymCopula.R mixCopula.R rotCopula.R
Encoding:	UTF-8
URL:	http://copula.r-forge.r-project.org/

Index of help topics:

```
.pairsCond              Pairs Plot of a cu.u Object (Internal Use)
A..Z                    Sinc, Zolotarev's, and Other Mathematical
                        Utility Functions
```

```
                         Confidence Intervals
rAntitheticVariates      Variance-Reduction Methods
rF01Frank                Sample Univariate Distributions Involved in
                         Nested Frank and Joe Copulas
rFFrank                  Sampling Distribution F for Frank and Joe
radSymTest               Test of Exchangeability for a Bivariate Copula
rdj                      Daily Returns of Three Stocks in the Dow Jones
retstable                Sampling Exponentially Tilted Stable
                         Distributions
rlog                     Sampling Logarithmic Distributions
rnacModel                Random nacopula Model
rnacopula                Sampling Nested Archimedean Copulas
rnchild                  Sampling Child 'nacopula's
rotCopula                Construction and Class of Rotated aka Reflected
                         Copulas
rstable1                 Random numbers from (Skew) Stable Distributions
safeUroot                One-dimensional Root (Zero) Finding - Extra
                         "Safety" for Convenience
serialIndepTest          Serial Independence Test for Continuous Time
                         Series Via Empirical Copula
setTheta                 Specify the Parameter(s) of a Copula
show-methods             Methods for 'show()' in Package 'copula'
splom2-methods           Methods for Scatter Plot Matrix 'splom2' in
                         Package 'copula'
tau                      Dependence Measures for Bivariate Copulas
tauAMH                   Ali-Mikhail-Haq ("AMH")'s and Joe's Kendall's
                         Tau
uranium                  Uranium Exploration Dataset of Cook & Johnson
                         (1986)
wireframe2-methods       Perspective Plots - 'wireframe2' in Package
                         'copula'
xvCopula                 Model (copula) selection based on 'k'-fold
                         cross-validation
```

Further information is available in the following vignettes:

`AC_Liouville`	Archimedean Liouville Copulas (source)
`AR_Clayton`	MLE and Quantile Evaluation for a Clayton AR(1) Model with Student Marginals (source)
`GIG`	Generalized Inverse Gaussian Archimedean Copulas (source)
`NALC`	Nested Archimedean Lévy Copulas (source)
`copula_GARCH`	The Copula GARCH Model (source)
`dNAC`	Densities of Two-Level Nested Archimedean Copulas (source)
`logL_visualization`	Log-Likelihood Visualization for Archimedean Copulas (source)
`qrng`	Quasi-Random Numbers for Copula Models (source)
`tail_compatibility`	Copula Constructions for Tail-Dependence Matrices (source)
`wild_animals`	Wild Animals: Examples of Nonstandard Copulas (source)
`Frank-Rmpfr`	Numerically stable Frank Copulas via Multiprecision (Rmpfr) (source)

nacopula-pkg	Nested Archimedean Copulas Meet R (source)
rhoAMH-dilog	Beautiful Spearman's Rho for AMH Copula (source)

The **copula** package provides

- Classes (S4) of commonly used copulas including elliptical (normal and t; ellipCopula), Archimedean (Clayton, Gumbel, Frank, Joe, and Ali-Mikhail-Haq; ; archmCopula and acopula), extreme value (Gumbel, Husler-Reiss, Galambos, Tawn, and t-EV; evCopula), and other families (Plackett and Farlie-Gumbel-Morgenstern).
- Methods for density, distribution, random number generation (dCopula, pCopula and rCopula); bivariate dependence measures (rho, tau, etc), perspective and contour plots.
- Functions (and methods) for fitting copula models including variance estimates (fitCopula).
- Independence tests among random variables and vectors.
- Serial independence tests for univariate and multivariate continuous time series.
- Goodness-of-fit tests for copulas based on multipliers, and the parametric bootstrap, with several transformation options.
- Bivariate and multivariate tests of extreme-value dependence.
- Bivariate tests of exchangeability.

Now with former package **nacopula** for working with nested Archimedean copulas. Specifically,

- it provides procedures for computing function values and cube volumes (prob),
- characteristics such as Kendall's tau and tail dependence coefficients (via family objects, e.g., copGumbel),
- efficient sampling algorithms (rnacopula),
- various estimators and goodness-of-fit tests.
- The package also contains related univariate distributions and special functions such as the Sibuya distribution (Sibuya), the polylogarithm (polylog), Stirling and Eulerian numbers (Eulerian).

Further information is available in the following vignettes:

nacopula-pkg	Nested Archimedean Copulas Meet R (../doc/nacopula-pkg.pdf)
Frank-Rmpfr	Numerically Stable Frank via Multiprecision in R (../doc/Frank-Rmpfr)

For a list of exported functions, use `help(package = "copula")`.

References

Yan, J. (2007) Enjoy the Joy of Copulas: With a Package **copula**. *Journal of Statistical Software* **21**(4), 1–21. http://www.jstatsoft.org/v21/i04/.

Kojadinovic, I. and Yan, J. (2010). Modeling Multivariate Distributions with Continuous Margins Using the copula R Package. *Journal of Statistical Software* **34**(9), 1–20. http://www.jstatsoft.org/v34/i09/.

Hofert, M. and Mächler, M. (2011), Nested Archimedean Copulas Meet R: The nacopula Package., *Journal of Statistical Software* **39**(9), 1–20. http://www.jstatsoft.org/v39/i09/.

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

See Also

The following CRAN packages currently use ('depend on') **copula**: **CoClust**, **copulaedas**, **Depela**, **HAC**, **ipptoolbox**, **vines**.

Examples

```
## Some of the more important functions (and their examples) are

example(fitCopula)## fitting Copulas
example(fitMvdc)  ## fitting multivariate distributions via Copulas
example(nacopula) ## nested Archimedean Copulas

## Independence Tests:  These also draw a 'Dependogram':
example(indepTest)       ## Testing for Independence
example(serialIndepTest) ## Testing for Serial Independence
```

.pairsCond — *Pairs Plot of a cu.u Object (Internal Use)*

Description

`.pairsCond()` is an internal function for plotting the pairwise Rosenblatt transforms, i.e., the pairwise conditional distributions, as returned by `pairwiseCcop()`, via the principal function `pairsRosenblatt()`.

The intention is that `pairsRosenblatt()` be called, rather than this auxiliary function.

Usage

```
.pairsCond(gcu.u, panel = points, colList,
     col = par("col"), bg = par("bg"), labels, ...,
     text.panel = textPanel, label.pos = 0.5,
     cex.labels = NULL, font.labels = 1, gap = 0,
     axes = TRUE, panel.border = TRUE, key = TRUE,
     keyOpt = list(space= 2.5, width= 1.5, axis= TRUE,
                   rug.at= numeric(), title= NULL, line= 5),
     main = NULL, main.centered = FALSE,
    line.main = if(is.list(main)) 5/4*par("cex.main")* rev(seq_along(main)) else 2,
     sub = NULL, sub.centered = FALSE, line.sub = 4)
```

Arguments

`gcu.u`	(n,d,d)-array of pairwise Rosenblatt-transformed u's as returned by `pairwiseCcop()`.
`panel`	panel function, as for `pairs()`.
`colList`	list of colors and information as returned by `pairsColList()`.
`col`	*instead* of `colList`, specifying the points' color.
`bg`	*instead* of `colList`, specifying the constant background color.
`labels`	pairs() argument; can be missing (in which case a suitable default is chosen or can be "none" [or something else])
`...`	further arguments, as for `pairs`. These are passed to `panel()`, and `axis`, may also contain `font.main`, `cex.main`, and `adj`, for title adjustments; further, `oma` for modifying the default `par("oma")`.
`text.panel, label.pos, cex.labels, font.labels, gap`	see `pairs()`.
`axes`	logical indicating whether axes are drawn.
`panel.border`	logical indicating whether a border is drawn around the pairs (to mimic the behavior of `image()`).
`key`	logical indicating whether a color key is drawn.
`keyOpt`	a `list` of options for the color key; `space`: white space in height of characters in inch to specify the the distance of the key to the pairs plot. `width`: key width in height of characters in inch. `axis`: logical indicating whether an axis for the color key is drawn. `rug.at`: values where rugs are plotted at the key. `title`: key title. `line`: key placement (horizontal distance from color key in lines).
`main`	title
`main.centered`	logical indicating if the title should be centered or not; the default `FALSE` centers it according to the pairs plot, not the whole plotting region.
`line.main`	title placement (vertical distance from pairs plot in lines).
`sub`	sub-title
`sub.centered`	logical indicating if the sub-title should be centered or not; see `main.centered`.
`line.sub`	sub-title placement, see `line.main`.

Note

based on pairs.default() and filled.contour() from R-2.14.1 - used in Hofert and Maechler (2013)

Author(s)

Marius Hofert and Martin Maechler

See Also

`pairsRosenblatt()`, the prinicipal function, calling `.pairsCond()`.

absdPsiMC — *Absolute Value of Generator Derivatives via Monte Carlo*

Description

Computes the absolute values of the dth generator derivative $\psi^{(d)}$ via Monte Carlo simulation.

Usage

```
absdPsiMC(t, family, theta, degree = 1, n.MC,
          method = c("log", "direct", "pois.direct", "pois"),
          log = FALSE, is.log.t = FALSE)
```

Arguments

`t`	`numeric` vector of evaluation points.
`family`	Archimedean family (name or object).
`theta`	parameter value.
`degree`	order d of the derivative.
`n.MC`	Monte Carlo sample size.
`method`	different methods: **`"log"`:** evaluates the logarithm of the sum involved in the Monte Carlo approximation in a numerically stable way; **`"direct"`:** directly evaluates the sum; **`"pois.direct"`:** interprets the sum in terms of the density of a Poisson distribution and evaluates this density directly; **`"pois"`:** as for `method="pois"` but evaluates the logarithm of the Poisson density in a numerically stable way.
`log`	if TRUE the logarithm of absdPsi is returned.
`is.log.t`	if TRUE the argument `t` contains the logarithm of the "mathematical" t, i.e., conceptually, `psi(t, *) == psi(log(t), *, is.log.t=TRUE)`, where the latter may potentially be numerically accurate, e.g., for $t = 10^{500}$, where as the former would just return $psi(Inf, *) = 0$.

Details

The absolute value of the dth derivative of the Laplace-Stieltjes transform $\psi = \mathcal{LS}[F]$ can be approximated via

$$(-1)^d \psi^{(d)}(t) = \int_0^\infty x^d \exp(-tx)\, dF(x) \approx \frac{1}{N} \sum_{k=1}^{N} V_k^d \exp(-V_k t),\ t > 0,$$

where $V_k \sim F,\ k \in \{1, \dots, N\}$. This approximation is used where $d =$`degree` and $N =$`n.MC`. Note that this is comparably fast even if `t` contains many evaluation points, since the random variates $V_k \sim F,\ k \in \{1, \dots, N\}$ only have to be generated once, not depending on `t`.

Value

numeric vector of the same length as t containing the absolute values of the generator derivatives.

References

Hofert, M., Mächler, M., and McNeil, A. J. (2013). Archimedean Copulas in High Dimensions: Estimators and Numerical Challenges Motivated by Financial Applications. *Journal de la Société Française de Statistique* **154**(1), 25–63.

See Also

acopula-families.

Examples

```
t <- c(0:100,Inf)
set.seed(1)
(ps <- absdPsiMC(t, family="Gumbel", theta=2, degree=10, n.MC=10000, log=TRUE))
## Note: The absolute value of the derivative at 0 should be Inf for
## Gumbel, however, it is always finite for the Monte Carlo approximation
set.seed(1)
ps2 <- absdPsiMC(log(t), family="Gumbel", theta=2, degree=10,
                 n.MC=10000, log=TRUE, is.log.t = TRUE)
stopifnot(all.equal(ps[-1], ps2[-1], tolerance=1e-14))
## Now is there an advantage of using "is.log.t" ?
sapply(eval(formals(absdPsiMC)$method), function(MM)
       absdPsiMC(780, family="Gumbel", method = MM,
                 theta=2, degree=10, n.MC=10000, log=TRUE, is.log.t = TRUE))
## not really better, yet...
```

---

acopula-class — *Class "acopula" of Archimedean Copula Families*

Description

This class "acopula" of Archimedean Copula Families is mainly used for providing objects of known Archimedean families with all related functions.

Objects from the Class

Objects can be created by calls of the form new("acopula", ...). For several well-known Archimedean copula families, the package **copula** already provides such family objects.

Slots

name: A string (class "character") describing the copula family, for example, "AMH" (or simply "A"), "Clayton" ("C"), "Frank" ("F"), "Gumbel" ("G"), or "Joe" ("J").

theta: Parameter value, a numeric, where NA means "unspecified".

psi, iPsi: The (Archimedean) generator ψ (with ψ(t)=exp(-t) being the generator of the independence copula) and its inverse (function). iPsi has an optional argument log which, if TRUE returns the logarithm of inverse of the generator.

absdPsi: A function which computes the absolute value of the derivative of the generator ψ for the given parameter theta and of the given degree (defaults to 1). Note that there is no informational loss by computing the absolute value since the derivatives alternate in sign (the generator derivative is simply (-1)^degree*absdPsi). The number n.MC denotes the sample size for a Monte Carlo evaluation approach. If n.MC is zero (the default), the generator derivatives are evaluated with their exact formulas. The optional parameter log (defaults to FALSE) indicates whether or not the logarithmic value is returned.

absdiPsi: a function computing the absolute value of the derivative of the generator inverse (iPsi()) for the given parameter theta. The optional parameter log (defaults to FALSE) indicates whether the logarithm of the absolute value of the first derivative of iPsi() is returned.

dDiag: a function computing the density of the diagonal of the Archimedean copula at u with parameter theta. The parameter log is as described before.

dacopula: a function computing the density of the Archimedean copula at u with parameter theta. The meanings of the parameters n.MC and log are as described before.

score: a function computing the *derivative* of the density with respect to the parameter θ.

uscore: a function computing the *derivative* of the density with respect to the each of the arguments.

paraInterval: Either NULL or an object of class "interval", which is typically obtained from a call such as interval("[a,b)").

paraConstr: A function of theta returning TRUE if and only if theta is a valid parameter value. Note that paraConstr is built automatically from the interval, whenever the paraInterval slot is valid. "interval".

nestConstr: A function, which returns TRUE if and only if the two provided parameters theta0 and theta1 satisfy the sufficient nesting condition for this family.

V0: A function which samples n random variates from the distribution F with Laplace-Stieltjes transform ψ and parameter theta.

dV0: A function which computes either the probability mass function or the probability density function (depending on the Archimedean family) of the distribution function whose Laplace-Stieltjes transform equals the generator ψ at the argument x (possibly a vector) for the given parameter theta. An optional argument log indicates whether the logarithm of the mass or density is computed (defaults to FALSE).

V01: A function which gets a vector of realizations of V0, two parameters theta0 and theta1 which satisfy the sufficient nesting condition, and which returns a vector of the same length as V0 with random variates from the distribution function F_{01} with Laplace-Stieltjes transform ψ_{01} (see dV01) and parameters $\theta_0 =$ theta0, $\theta_1 =$ theta1.

dV01: Similar to dV0 with the difference being that this function computes the probability mass or density function for the Laplace-Stieltjes transform

$$\psi_{01}(t; V_0) = \exp(-V_0\psi_0^{-1}(\psi_1(t))),$$

corresponding to the distribution function F_{01}.

Arguments are the evaluation point(s) x, the value(s) V0, and the parameters theta0 and theta1. As for dV0, the optional argument log can be specified (defaults to FALSE). Note that if x is a vector, V0 must either have length one (in which case V0 is the same for every component of x) or V0 must be of the same length as x (in which case the components of V0 correspond to the ones of x).

tau, iTau: Compute Kendall's tau of the bivariate Archimedean copula with generator ψ as a function of theta, respectively, theta as a function of Kendall's tau.

lambdaL, lambdaU, lambdaLInv, lambdaUInv: Compute the lower (upper) tail-dependence coefficient of the bivariate Archimedean copula with generator ψ as a function of theta, respectively, theta as a function of the lower (upper) tail-dependence coefficient.

For more details about Archimedean families, corresponding distributions and properties, see the references.

Methods

initialize signature(.Object = "acopula"): is used to automatically construct the function slot paraConstr, when the paraInterval is provided (typically via interval()).

show signature("acopula"): compact overview of the copula.

References

See those of the families, for example, copGumbel.

See Also

Specific provided copula family objects, for example, copAMH, copClayton, copFrank, copGumbel, copJoe.
To access these, you may also use getAcop.

A *nested* Archimedean copula *without* child copulas (see class "nacopula") is a proper Archimedean copula, and hence, onacopula() can be used to construct a specific parametrized Archimedean copula; see the example below.

Alternatively, setTheta also returns such a (parametrized) Archimedean copula.

Examples

```
## acopula class information
showClass("acopula")

## Information and structure of Clayton copulas
copClayton
str(copClayton)
```

```
## What are admissible parameters for Clayton copulas?
copClayton@paraInterval

## A Clayton "acopula" with Kendall's tau = 0.8 :
(cCl.2 <- setTheta(copClayton, iTau(copClayton, 0.8)))

## Can two Clayton copulas with parameters theta0 and theta1 be nested?
## Case 1: theta0 = 3, theta1 = 2
copClayton@nestConstr(theta0 = 3, theta1 = 2)
## -> FALSE as the sufficient nesting criterion is not fulfilled
## Case 2: theta0 = 2, theta1 = 3
copClayton@nestConstr(theta0 = 2, theta1 = 3) # TRUE

## For more examples, see  help("acopula-families")
```

---

acR *Distribution of the Radial Part of an Archimedean Copula*

Description

pacR() computes the distribution function F_R of the radial part of an Archimedean copula, given by

$$F_R(x) = 1 - \sum_{k=0}^{d-1} \frac{(-x)^k \psi^{(k)}(x)}{k!},\ x \in [0, \infty);$$

The formula (in a slightly more general form) is given by McNeil and G. Nešlehová (2009).

qacR() computes the quantile function of F_R.

Usage

```
pacR(x, family, theta, d, lower.tail = TRUE, log.p = FALSE, ...)
qacR(p, family, theta, d, log.p = FALSE, interval,
     tol = .Machine$double.eps^0.25, maxiter = 1000, ...)
```

Arguments

x	numeric vector of nonnegative evaluation points for F_R.
p	numeric vector of evaluation points of the quantile function.
family	Archimedean family.
theta	parameter $theta$.
d	dimension d.
lower.tail	logical; if TRUE, probabilities are $P[X <= x]$ otherwise, $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as $\log p$.
interval	root-search interval.
tol	see uniroot().
maxiter	see uniroot().
...	additional arguments passed to the procedure for computing derivatives.

Value

The distribution function of the radial part evaluated at x, or its inverse, the quantile at p.

References

McNeil, A. J., G. Nešlehová, J. (2009). Multivariate Archimedean copulas, d-monotone functions and l_1-norm symmetric distributions. *The Annals of Statistics* **37**(5b), 3059–3097.

Examples

```
## setup
family <- "Gumbel"
tau <- 0.5
m <- 256
dmax <- 20
x <- seq(0, 20, length.out=m)

## compute and plot pacR() for various d's
y <- vapply(1:dmax, function(d)
            pacR(x, family=family, theta=iTau(archmCopula(family), tau), d=d),
            rep(NA_real_, m))
plot(x, y[,1], type="l", ylim=c(0,1),
     xlab = quote(italic(x)), ylab = quote(F[R](x)),
     main = substitute(italic(F[R](x))~~ "for" ~ d==1:.D, list(.D = dmax)))
for(k in 2:dmax) lines(x, y[,k])
```

---

allComp *All Components of a (Inner or Outer) Nested Archimedean Copula*

Description

Given the nested Archimedean copula x, return an integer vector of the *indices* of all components of the corresponding outer_nacopula which are components of x, either direct components or components of possible child copulas. This is typically only used by programmers investigating the exact nesting structure.

For an outer_nacopula object x, allComp(x) must be the same as 1:dim(x), whereas its “inner” component copulas will each contain a *subset* of those indices only.

Usage

```
allComp(x)
```

Arguments

x	an R object inheriting from class nacopula.

Value

An integer vector of indices j of all components u_j as described in the description above.

Examples

```
C3 <- onacopula("AMH", C(0.7135, 1, C(0.943, 2:3)))
allComp(C3) # components are 1:3
allComp(C3@childCops[[1]]) # for the child, only  (2, 3)
```

An *Nonparametric Rank-based Estimators of the Pickands Dependence Function*

Description

Bivariate and multivariate versions of the nonparametric rank-based estimators of the Pickands dependence function A, studied in Genest and Segers (2009) and Gudendorf and Segers (2011).

Usage

```
An.biv(x, w, estimator = c("CFG", "Pickands"), corrected = TRUE,
       ties.method = eval(formals(rank)$ties.method))
An(x, w, ties.method = eval(formals(rank)$ties.method))
```

Arguments

`x`	a data matrix that will be transformed to pseudo-observations. If `An.biv` is called, `x` has to have two columns.
`w`	if `An.biv` is called, a vector of points in [0,1] where to evaluate the estimated bivariate Pickands dependence function. If the multivariate estimator `An` is used instead, `w` needs to be a matrix with the same number of columns as `x` whose lines are elements of the multivariate unit simplex (see the last reference).
`estimator`	specifies which nonparametric rank-based estimator of the unknown Pickands dependence function to use in the bivariate case; can be either `"CFG"`(Capéraà-Fougères-Genest) or `"Pickands"`.
`corrected`	TRUE means that the bivariate estimators will be corrected to ensure that their value at 0 and 1 is 1.
`ties.method`	`character` string specifying how ranks should be computed if there are ties in any of the coordinate samples of `x`; passed to `pobs`.

Details

More details can be found in the references.

Value

`An.biv()` returns a vector containing the values of the estimated Pickands dependence function at the points in `w` (and is the same as former `Anfun()`).

The function `An` computes simultaneously the three corrected multivariate estimators studied in Gudendorf and Segers (2011) at the points in `w` and retuns a list whose components are

P	values of the Pickands estimator at the points in w.
CFG	values of the CFG estimator at the points in w.
HT	values of the Hall-Tajvidi estimator at the points in w.

References

C. Genest and J. Segers (2009). Rank-based inference for bivariate extreme-value copulas. *Annals of Statistics* **37**, 2990–3022.

G. Gudendorf and J. Segers (2011). Nonparametric estimation of multivariate extreme-value copulas. *Journal of Statistical Planning and Inference* **142**, 3073–3085.

See Also

evCopula, A, and evTestA. Further, evTestC, evTestK, exchEVTest, and gofEVCopula.

Examples

```
## True Pickands dependence functions
curve(A(gumbelCopula(4   ), x), 0, 1)
curve(A(gumbelCopula(2   ), x), add=TRUE, col=2)
curve(A(gumbelCopula(1.33), x), add=TRUE, col=3)

## CFG estimator
curve(An.biv(rCopula(1000, gumbelCopula(4   )), x), lty=2, add=TRUE)
curve(An.biv(rCopula(1000, gumbelCopula(2   )), x), lty=2, add=TRUE, col=2)
curve(An.biv(rCopula(1000, gumbelCopula(1.33)), x), lty=2, add=TRUE, col=3)

## Pickands estimator
curve(An.biv(rCopula(1000, gumbelCopula(4   )), x, estimator="Pickands"),
      lty=3, add=TRUE)
curve(An.biv(rCopula(1000, gumbelCopula(2   )), x, estimator="Pickands"),
      lty=3, add=TRUE, col=2)
curve(An.biv(rCopula(1000, gumbelCopula(1.33)), x, estimator="Pickands"),
      lty=3, add=TRUE, col=3)

legend("bottomleft",  paste0("Gumbel(", format(c(4, 2, 1.33)),")"),
       lwd=1, col=1:3, bty="n")
legend("bottomright", c("true", "CFG est.", "Pickands est."), lty=1:3, bty="n")

## Relationship between An.biv and An
u <- c(runif(100),0,1) # include 0 and 1
x <- rCopula(1000, gumbelCopula(4))
r <- An(x, cbind(1-u, u))
all.equal(r$CFG, An.biv(x, u))
all.equal(r$P, An.biv(x, u, estimator="Pickands"))

## A trivariate example
x <- rCopula(1000, gumbelCopula(4, dim = 3))
u <- matrix(runif(300), 100, 3)
w <- u / apply(u, 1, sum)
r <- An(x, w)
```

```
## Endpoint corrections are applied
An(x, cbind(1, 0, 0))
An(x, cbind(0, 1, 0))
An(x, cbind(0, 0, 1))
```

---

archmCopula *Construction of Archimedean Copula Class Object*

Description

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

Usage

```
archmCopula(family, param = NA_real_, dim = 2, ...)

claytonCopula(param = NA_real_, dim = 2,
          use.indepC = c("message", "TRUE", "FALSE"))
frankCopula(param = NA_real_, dim = 2,
          use.indepC = c("message", "TRUE", "FALSE"))
gumbelCopula(param = NA_real_, dim = 2,
          use.indepC = c("message", "TRUE", "FALSE"))
amhCopula(param = NA_real_, dim = 2,
          use.indepC = c("message", "TRUE", "FALSE"))
joeCopula(param = NA_real_, dim = 2,
          use.indepC = c("message", "TRUE", "FALSE"))
```

Arguments

`family`	a character string specifying the family of an Archimedean copula. Currently supported families are "clayton", "frank", "amh", "gumbel", and "joe".
`param`	number (`numeric`) specifying the copula parameter.
`dim`	the dimension of the copula.
`...`	further arguments, passed to the individual creator functions (`claytonCopula()`, etc).
`use.indepC`	a string specifying if the independence copula `indepCopula`, should be returned in the case where the parameter θ, `param`, is at the boundary or limit case where the corresponding Archimedean copula is the independence copula. The default does return `indepCopula()` with a message, using `"TRUE"` does it without a message. This makes the resulting object more useful typically, but does not return a formal Archimedean copula of the desired family, something needed e.g., for fitting purposes, where you'd use `use.indepC="FALSE"`.

Details

`archmCopula()` is a wrapper for `claytonCopula()`, `frankCopula()`, `gumbelCopula()`, `amhCopula()` and `joeCopula`.

For the mathematical definitions of the respective Archimedean families, see `copClayton`.

For $d = 2$, i.e. `dim = 2`, the AMH, Clayton and Frank copulas allow to model negative Kendall's tau (`tau`) behavior via negative θ, for AMH and Clayton $-1 \leq \theta$, and for Frank $-\infty < \theta$. The respective boundary cases are

AMH: $\tau(\theta = -1) = -0.1817258$,

Clayton: $\tau(\theta = -1) = -1$,

Frank: $\tau(\theta = -Inf) = -1$ (as limit).

For the Ali-Mikhail-Haq copula family (`"amhCopula"`), only the bivariate case is available; however `copAMH` has no such limitation.

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

For $d > 2$, i.e. `dim > 2`, it is now recommended to work with the `acopula`-classed Archimedean copulas, as there is no restriction on the dimension there.

Value

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

References

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

See Also

`acopula`-classed Archimedean copulas, such as `copClayton`, `copGumbel`, etc, notably for mathematical definitions including the meaning of `param`.

`fitCopula` for fitting such copulas to data.

`ellipCopula`, `evCopula`.

Examples

```
clayton.cop <- claytonCopula(2, dim = 3)
## scatterplot3d(rCopula(1000, clayton.cop))

## negative param (= theta) is allowed for dim = 2 :
tau(claytonCopula(-0.5)) ## = -1/3
tauClayton <- Vectorize(function(theta) tau(claytonCopula(theta, dim=2)))
plot(tauClayton, -1, 10, xlab=quote(theta), ylim = c(-1,1), n=1025)
abline(h=-1:1,v=0, col="#11111150", lty=2); axis(1, at=-1)

tauFrank <- Vectorize(function(theta) tau(frankCopula(theta, dim=2)))
plot(tauFrank, -40, 50, xlab=quote(theta), ylim = c(-1,1), n=1025)
```

```
abline(h=-1:1,v=0, col="#11111150", lty=2)

## tauAMH() is function in our package
iTau(amhCopula(), -1) # -1 with a range warning
iTau(amhCopula(), (5 - 8*log(2)) / 3) # -1 with a range warning

ic <- frankCopula(0) # independence copula (with a "message")
stopifnot(identical(ic,
   frankCopula(0, use.indepC = "TRUE")))# indep.copula  withOUT message
(fC <- frankCopula(0, use.indepC = "FALSE"))
## a Frank copula which corresponds to the indep.copula (but is not)

frankCopula(dim = 3)# with NA parameters
frank.cop <- frankCopula(3)# dim=2
persp(frank.cop, dCopula)

gumbel.cop <- archmCopula("gumbel", 5)
stopifnot(identical(gumbel.cop, gumbelCopula(5)))
contour(gumbel.cop, dCopula)

amh.cop <- amhCopula(0.5)
u. <- as.matrix(expand.grid(u=(0:10)/10, v=(0:10)/10, KEEP.OUT.ATTRS=FALSE))
du <- dCopula(u., amh.cop)
stopifnot(is.finite(du) | apply(u. == 0, 1,any)| apply(u. == 1, 1,any))

## A 7-dim Frank copula
frank.cop <- frankCopula(3, dim = 7)
x <- rCopula(5, frank.cop)
## dCopula now *does* work:
dCopula(x, frank.cop)

## A 7-dim Gumbel copula
gumbelC.7 <- gumbelCopula(2, dim = 7)
dCopula(x, gumbelC.7)

## A 12-dim Joe copula
joe.cop <- joeCopula(iTau(joeCopula(), 0.5), dim = 12)
x <- rCopula(5, joe.cop)
dCopula(x, joe.cop)
```

---

archmCopula-class *Class "archmCopula"*

Description

Archimedean copula class.

Objects from the Class

Created by calls of the form `new("archmCopula", ...)` or rather typically by `archmCopula()`. Implemented families are Clayton, Gumbel, Frank, Joe, 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, parameters, etc: all inherited from the super class 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"): ...

dCopula signature(copula = "joeCopula"): ...

pCopula signature(copula = "joeCopula"): ...

rCopula signature(copula = "joeCopula"): ...

Extends

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

Note

"gumbelCopula" is also of class "evCopula".

See Also

archmCopula, for constructing such copula objects; copula-class.

`assocMeasures` *Dependence Measures for Bivariate Copulas*

Description

These functions compute Kendall's tau, Spearman's rho, and the tail dependence index for *bivariate* copulas. `iTau` and `iRho`, sometimes called "calibration" functions are the inverses: they determine ("calibrate") the copula parameter (which must be one-dimensional!) given the value of Kendall's tau or Spearman's rho.

Usage

```
tau (copula, ...)
rho (copula, ...)
lambda(copula, ...)
iTau (copula, tau, ...)
iRho (copula, rho, ...)
```

Arguments

`copula`	an R object of class "`copula`" (or also "`acopula`" or "`nacopula`"; note however that some methods may not be available for some copula families).
`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 functions `iTau()` and `iRho()` in fact return 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.

Iwan Kojadinovic and Jun Yan (2010) Comparison of three semiparametric methods for estimating dependence parameters in copula models. *Insurance: Mathematics and Economics* **47**, 52–63.

See Also

The `acopula` class objects have slots, `tau`, `lambdaL`, and `lambdaU` providing functions for `tau()`, and the two tail indices `lambda()`, and slot `iTau` for `iTau()`, see the examples and `copGumbel`, etc.

Examples

```
gumbel.cop <- gumbelCopula(3)
tau(gumbel.cop)
rho(gumbel.cop)
lambda(gumbel.cop)
iTau(joeCopula(), 0.5)

stopifnot(all.equal(tau(gumbel.cop), copGumbel@tau(3)),

          all.equal(lambda(gumbel.cop),
                    c(copGumbel@lambdaL(3), copGumbel@lambdaU(3)),
                    check.attributes=FALSE),

          all.equal(iTau (gumbel.cop, 0.681),
                    copGumbel@iTau(0.681))
)

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

Bernoulli *Compute Bernoulli Numbers*

Description

Compute the nth Bernoulli number, or generate all Bernoulli numbers up to the nth, using diverse methods, that is, algorithms.

NOTE the current default methods will be changed – to get better accuracy!

Usage

```
Bernoulli    (n, method = c("sumBin", "sumRamanujan", "asymptotic"),
              verbose = FALSE)
Bernoulli.all(n, method = c("A-T", "sumBin", "sumRamanujan", "asymptotic"),
              precBits = NULL, verbose = getOption("verbose"))
```

Arguments

n — positive integer, indicating the index of the largest (and last) of the Bernoulli numbers needed.

method — character string, specifying which method should be applied. The default for Bernoulli.all(), "A-T" stands for the Akiyama-Tanigawa algorithm which is nice and simple but has bad numerical properties. It can however work with high precision "mpfr"-numbers, see `precBits`. `"sumRamanujan"` is somewhat more efficient but not yet implemented.

`precBits` currently only for `method = "A-T"` – `NULL` or a positive integer indicating the precision of the initial numbrs in bits, using **"Rmpfr"**'s package multiprecision arithmetic.

`verbose` (for `"A-T"`:) logical indicating if the intermediate results of the algorithm should be printed.

Value

`Bernoulli()`**:** a number

`Bernoulli.all()`**:** a numeric vector of length n, containing B(n)

References

Kaneko, Masanobu (2000) The Akiyama-Tanigawa algorithm for Bernoulli numbers; Journal of Integer Sequences **3**, article 00.2.9

See Also

`Eulerian`, `Stirling1`, etc.

Examples

```
## The example for the paper
MASS::fractions(Bernoulli.all(8, verbose=TRUE))

B10 <- Bernoulli.all(10)
MASS::fractions(B10)

system.time(B50  <- Bernoulli.all(50))#  {does not cache} -- still "no time"
system.time(B100 <- Bernoulli.all(100))# still less than a milli second

## Using Bernoulli() is not much slower, but hopefully *more* accurate!
## Check first - TODO
system.time(B.1c <- Bernoulli(100))# caches ..
system.time(B1c. <- Bernoulli(100))# ==> now much faster
stopifnot(identical(B.1c, B1c.))

if(FALSE)## reset the cache:
assign("Bern.tab", list(), envir = copula:::.nacopEnv)

## More experiments in the source of the copula package ../tests/Stirling-etc.R
```

`beta.Blomqvist` *Sample and Population Version of Blomqvist's Beta for Archimedean Copulas*

Description

Compute the population (`beta.()`) and sample (`betan()`) version of Blomqvist's beta for an Archimedean copula.

See the reference below for definitions and formulas.

Usage

```
beta.(cop, theta, d, scaling=FALSE)
betan(u, scaling=FALSE)
```

Arguments

`cop`	an Archimedean copula (of dimension d) to be estimated.
`theta`	copula parameter.
`d`	dimension.
`scaling`	logical, if true, the factors 2^(d-1)/(2^(d-1)-1) and 2^(1-d) in Blomqvist's beta are omitted.
`u`	For `betan`: $(n \times d)$-matrix of d-dimensional observations distributed according to the copula.

Value

`beta.`: a number, being the population version of Blomqvist's beta for the corresponding Archimedean copula;

`betan`: a number, being the sample version of Blomqvist's beta for the given data.

References

Schmid and Schmidt (2007), Nonparametric inference on multivariate versions of Blomqvist's beta and related measures of tail dependence, *Metrika* **66**, 323–354.

See Also

`acopula`

Examples

```
beta.(copGumbel, 2.5, d = 5)

d.set <- c(2:6, 8, 10, 15, 20, 30)
cols <- adjustcolor(colorRampPalette(c("red", "orange", "blue"),
                                     space = "Lab")(length(d.set)), 0.8)
## AMH:
for(i in seq_along(d.set))
   curve(Vectorize(beta.,"theta")(copAMH, x, d = d.set[i]), 0, .999999,
         main = "Blomqvist's beta(.) for  AMH",
         xlab = quote(theta), ylab = quote(beta(theta, AMH)),
         add = (i > 1), lwd=2, col=cols[i])
mtext("NB:  d=2 and d=3 are the same")
legend("topleft", paste("d =",d.set), bty="n", lwd=2, col=cols)

## Gumbel:
for(i in seq_along(d.set))
   curve(Vectorize(beta.,"theta")(copGumbel, x, d = d.set[i]), 1, 10,
         main = "Blomqvist's beta(.) for  Gumbel",
         xlab = quote(theta), ylab = quote(beta(theta, Gumbel)),
         add=(i > 1), lwd=2, col=cols[i])
legend("bottomright", paste("d =",d.set), bty="n", lwd=2, col=cols)

## Clayton:
for(i in seq_along(d.set))
   curve(Vectorize(beta.,"theta")(copClayton, x, d = d.set[i]), 1e-5, 10,
         main = "Blomqvist's beta(.) for  Clayton",
         xlab = quote(theta), ylab = quote(beta(theta, Gumbel)),
         add=(i > 1), lwd=2, col=cols[i])
legend("bottomright", paste("d =",d.set), bty="n", lwd=2, col=cols)

## Joe:
for(i in seq_along(d.set))
   curve(Vectorize(beta.,"theta")(copJoe, x, d = d.set[i]), 1, 10,
         main = "Blomqvist's beta(.) for  Joe",
         xlab = quote(theta), ylab = quote(beta(theta, Gumbel)),
         add=(i > 1), lwd=2, col=cols[i])
legend("bottomright", paste("d =",d.set), bty="n", lwd=2, col=cols)

## Frank:
for(i in seq_along(d.set))
   curve(Vectorize(beta.,"theta")(copFrank, x, d = d.set[i]), 1e-5, 50,
         main = "Blomqvist's beta(.) for  Frank",
         xlab = quote(theta), ylab = quote(beta(theta, Gumbel)),
         add=(i > 1), lwd=2, col=cols[i])
legend("bottomright", paste("d =",d.set), bty="n", lwd=2, col=cols)

## Shows the numeric problems:
curve(Vectorize(beta.,"theta")(copFrank, x, d = 29), 35, 42, col="violet")
```

C.n *The Empirical Copula*

Description

Given pseudo-observations from a distribution with continuous margins and copula C, the *empirical copula* is the empirical distribution function of these pseudo-observations. It is thus a natural nonparametric estimator of C. The function `C.n()` computes the empirical copula or two alternative smoothed versions of the latter: the *empirical beta copula* or the *empirical checkerboard copula*; see Eqs. (2.1) and (4.1) in Segers, Sibuya and Tsukahara (2017), and the references therein.

The function `dCn()` approximates first-order partial derivatives of the unknown copula using the empirical copula.

The function `F.n()` computes the empirical distribution function of a multivariate sample. Note that `C.n(u, X, smoothing="none", *)` simply calls `F.n(u, pobs(X), *)` after checking `u`.

Usage

```
C.n(u, X, smoothing = c("none", "beta", "checkerboard"),
    offset = 0, method = c("C", "R"),
    ties.method = c("max", "average", "first", "last", "random", "min"))
dCn(u, U, j.ind=1:d, b=1/sqrt(nrow(U)), ...)

F.n(x, X, offset=0, method=c("C", "R"))

Cn(x, w) ## <-- deprecated!  use  C.n(w, x) instead!
```

Arguments

`u,w`	an (m, d)-`matrix` with elements in $[0, 1]$ whose rows contain the evaluation points of the empirical copula.
`x`	an (m, d)-`matrix` whose rows contain the evaluation points of the empirical distribution function.
`U`	for `dCN()` only: an (n, d)-`matrix` with elements in $[0, 1]$ and with the same number d of columns as `u`. The rows of `U` are the pseudo-observations based on which the empirical copula is computed.
`X`	(and `x` and `U` for `Cn()`:) an (n, d)-`matrix` with the same number d of columns as `x`. Recall that a multivariate random sample `X` can be transformed to an appropriate `U` via `pobs()`.
`smoothing`	`character` string specifying whether the empirical copula (`smoothing="none"`), the empirical beta copula (`smoothing="beta"`) or the empirical checkerboard copula (`smoothing="checkerboard"`) is computed.
`ties.method`	`character` string specifying how ranks should be computed if there are ties in any of the coordinate samples of `x`; passed to `pobs`.
`j.ind`	`integer` vector of indices j between 1 and d indicating the dimensions with respect to which first-order partial derivatives are approximated.

`b`	`numeric` giving the bandwidth for approximating first-order partial derivatives.
`offset`	used in scaling the result which is of the form `sum(....)/(n+offset)`; defaults to zero.
`method`	`character` string indicating which method is applied to compute the empirical cumulative distribution function or the empirical copula. `method="C"` uses a an implementation in C, `method="R"` uses a pure R implementation.
`...`	additional arguments passed to `dCn()`.

Details

There are several asymptotically equivalent definitions of the empirical copula. As mentioned above, the empirical copula `C.n(, smoothing = "none")` is simply defined as the empirical distribution function computed from the pseudo-observations, that is,

$$C_n(\boldsymbol{u}) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{\hat{\boldsymbol{U}}_i \le \boldsymbol{u}\}},$$

where $\hat{\boldsymbol{U}}_i$, $i \in \{1, \ldots, n\}$, denote the pseudo-observations (rows in `U`) and n the sample size. Internally, `C.n(,smoothing = "none")` is just a wrapper for `F.n()` and is expected to be fed with the pseudo-observations.

The approximation for the jth partial derivative of the unknown copula C is implemented as, for example, in Rémillard and Scaillet (2009), and given by

$$\hat{\dot{C}}_{jn}(\boldsymbol{u}) = \frac{C_n(u_1, .., u_{j-1}, min(u_j + b, 1), u_{j+1}, .., u_d) - C_n(u_1, .., u_{j-1}, max(u_j - b, 0), u_{j+1}, .., u_d)}{2b},$$

where b denotes the bandwidth and C_n the empirical copula.

Value

`C.n()` returns the empirical copula at `u` or a smoothed version of the latter.

`F.n()` returns the empirical distribution function of `X` evaluated at `x`.

`dCn()` returns a vector (`length(j.ind)` is 1) or a matrix (with number of columns equal to `length(j.ind)`), containing the approximated first-order partial derivatives of the unknown copula at `u` with respect to the arguments in `j.ind`.

Note

The first version of our empirical copula implementation, `Cn()`, had its two arguments *reversed* compared to `C.n()`, and is deprecated now. You **must** swap its arguments to transform to new code.

The use of the two smoothed versions assumes implicitly no ties in the component samples of the data.

References

Rüschendorf, L. (1976). Asymptotic distributions of multivariate rank order statistics, *Annals of Statistics* **4**, 912–923.

Deheuvels, P. (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.

Deheuvels, P. (1981). A non parametric test for independence, *Publ. Inst. Statist. Univ. Paris* **26**, 29–50.

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

Segers, J., Sibuya, M. and Tsukahara, H. (2017). The Empirical Beta Copula. *Journal of Multivariate Analysis*, 155, pages 35–51, http://arxiv.org/abs/1607.04430.

See Also

`pobs()` for computing pseudo-observations, `pCopula()` for evaluating a copula.

Examples

```
## Generate data X (from a meta-Gumbel model with N(0,1) margins)
n <- 100
d <- 3
family <- "Gumbel"
theta <- 2
cop <- onacopulaL(family, list(theta=theta, 1:d))
set.seed(1)
X <- qnorm(rCopula(n, cop)) # meta-Gumbel data with N(0,1) margins

## Random points were to evaluate the empirical copula
u <- matrix(runif(n*d), n, d)
ec <- C.n(u, X)

## Compare the empirical copula with the true copula
pc <- pCopula(u, copula=cop)
mean(abs(pc - ec)) # ~= 0.012 -- increase n to decrease this error

## The same for the two smoothed versions
beta <- C.n(u, X, smoothing = "beta")
mean(abs(pc - beta))
check <- C.n(u, X, smoothing = "checkerboard")
mean(abs(pc - check))

## Compare the empirical copula with F.n(pobs())
U <- pobs(X) # pseudo-observations
stopifnot(identical(ec, F.n(u, X=pobs(U)))) # even identical

## Compare the empirical copula based on U at U with the Kendall distribution
## Note: Theoretically, C(U) ~ K, so K(C_n(U, U=U)) should approximately be U(0,1)
plot(pK(C.n(U, X), cop=cop@copula, d=d))

## Compare the empirical copula and the true copula on the diagonal
C.n.diag <- function(u) C.n(do.call(cbind, rep(list(u), d)), X=X) # diagonal of C_n
C.diag <- function(u) pCopula(do.call(cbind, rep(list(u), d)), cop) # diagonal of C
curve(C.n.diag, from=0, to=1, # empirical copula diagonal
      main=paste("True vs empirical diagonal of a", family, "copula"),
```

```
      xlab="u", ylab=quote("True C(u,..,u) and empirical"~C[n](u,..,u)))
curve(C.diag, lty=2, add=TRUE) # add true copula diagonal
legend("bottomright", lty=2:1, bty="n", inset=0.02,
       legend = expression(C, C[n]))

## Approximate partial derivatives w.r.t. the 2nd and 3rd component
j.ind <- 2:3 # indices w.r.t. which the partial derivatives are computed
## Partial derivatives based on the empirical copula and the true copula
der23 <- dCn(u, U=pobs(U), j.ind=j.ind)
der23. <- copula:::dCdu(archmCopula(family, param=theta, dim=d), u=u)[,j.ind]
## Approximation error
summary(as.vector(abs(der23-der23.)))

## For an example of using F.n(), see help(mvdc)% ./Mvdc.Rd
```

`cCopula` *Conditional Distributions and Their Inverses from Copulas*

Description

Compute the conditional distribution function $C(u_d \,|\, u_1, \dots, u_{d-1})$ of u_d given $u_1, \dots, u_{d-1}$.

Usage

```
cCopula(u, copula, indices = 1:dim(copula), inverse = FALSE,
        log = FALSE, drop = FALSE, ...)

## Deprecated (use cCopula() instead):
rtrafo(u, copula, indices = 1:dim(copula), inverse = FALSE, log = FALSE)
cacopula(u, cop, n.MC = 0, log = FALSE)
```

Arguments

`u`	data `matrix` in $[0,1]^(n,d)$ of $U(0,1)^d$ samples if `inverse = FALSE` and (pseudo-/copula-)observations if `inverse = TRUE`.
`copula, cop`	copula, i.e., an object of class "`Copula`" with specified parameters; currently, the conditional distribution is only provided for Archimedean and elliptical copulas.
`indices`	vector of indices j (in $\{1,\dots,d\}$ ($d =$ copula dimension); unique; sorted in increasing order) for which $C_{j\|1,\dots,j-1}(u_j \,\|\, u_1,\dots,u_{j-1})$ (or, if `inverse = TRUE`, $C^-_{j\|1,\dots,j-1}(u_j \,\|\, u_1,\dots,u_{j-1})$) is computed.
`inverse`	`logical` indicating whether the inverse $C^-_{j\|1,\dots,j-1}(u_j \,\|\, u_1,\dots,u_{j-1})$ is returned.
`n.MC`	integer Monte Carlo sample size; for Archimedean copulas only, used if positive.
`log`	a `logical` indicating whether the logarithmic values are returned.
`drop`	a `logical` indicating whether a vector should be returned (instead of a 1–row matrix) when `n` is 1.
`...`	additional arguments (currently only used if `inverse = TRUE` in which case they are passed on to the underlying `uniroot()`).

Details

By default and if fed with a sample of the corresponding copula, `cCopula()` computes the Rosenblatt transform; see Rosenblatt (1952). The involved high-order derivatives for Archimedean copulas were derived in Hofert et al. (2012).

Sampling, that is, random number generation, can be achieved by using `inverse=TRUE`. In this case, the inverse Rosenblatt transformation is used, which, for sampling purposes, is also known as *conditional distribution method*. Note that, for Archimedean copulas not being Clayton, this can be slow as it involves numerical root finding in each (but the first) component.

Value

An (n, k)-`matrix` (unless `n == 1` and `drop` is true, where a k-vector is returned) where k is the length of `indices`. This matrix contains the conditional copula function values $C_{j|1,\dots,j-1}(u_j \mid u_1, \dots, u_{j-1})$ or, if `inverse = TRUE`, their inverses $C^{-}_{j|1,\dots,j-1}(u_j \mid u_1, \dots, u_{j-1})$ for all j in `indices`.

Note

For some (but not all) families, this function also makes sense on the boundaries (if the corresponding limits can be computed).

References

Genest, C., Rémillard, B., and Beaudoin, D. (2009). Goodness-of-fit tests for copulas: A review and a power study. *Insurance: Mathematics and Economics* **44**, 199–213.

Rosenblatt, M. (1952). Remarks on a Multivariate Transformation, *The Annals of Mathematical Statistics* **23**, 3, 470–472.

Hofert, M., Mächler, M., and McNeil, A. J. (2012). Likelihood inference for Archimedean copulas in high dimensions under known margins. *Journal of Multivariate Analysis* **110**, 133–150.

See Also

`htrafo`; `acopula-families`.

Examples

```
## 1) Sampling from a conditional distribution of a Clayton copula given u_1

## Define the copula
tau <- 0.5
theta <- iTau(claytonCopula(), tau = tau)
d <- 2
cc <- claytonCopula(theta, dim = d)
n <- 1000
set.seed(271)

## A small u_1
u1 <- 0.05
U <- cCopula(cbind(u1, runif(n)), copula = cc, inverse = TRUE)
plot(U[,2], ylab = quote(U[2]))
```

```
## A large u_1
u1 <- 0.95
U <- cCopula(cbind(u1, runif(n)), copula = cc, inverse = TRUE)
plot(U[,2], ylab = quote(U[2]))


## 2) Sample via conditional distribution method and then apply the
##    Rosenblatt transform
##    Note: We choose the numerically more involved (and thus slower)
##          Gumbel case here

## Define the copula
tau <- 0.5
theta <- iTau(gumbelCopula(), tau = tau)
d <- 5
gc <- gumbelCopula(theta, dim = d)
n <- 200
set.seed(271)
U. <- matrix(runif(n*d), ncol = d) # U(0,1)^d


## Transform to Gumbel sample via conditional distribution method
U <- cCopula(U., copula = gc, inverse = TRUE) # slow for ACs except Clayton
splom2(U) # scatter-plot matrix copula sample

## Rosenblatt transform back to U(0,1)^d (as a check)
U. <- cCopula(U, copula = gc)
splom2(U.) # U(0,1)^d again


## 3) cCopula() for elliptical copulas

tau <- 0.5
theta <- iTau(claytonCopula(), tau = tau)
d <- 5
cc <- claytonCopula(theta, dim = d)
set.seed(271)
n <- 1000
U <- rCopula(n, copula = cc)
X <- qnorm(U) # X now follows a meta-Clayton model with N(0,1) marginals
U <- pobs(X) # build pseudo-observations

fN <- fitCopula(normalCopula(dim = d), data = U) # fit a Gauss copula
U.RN <- cCopula(U, copula = fN@copula)
splom2(U.RN, cex = 0.2) # visible but not so clearly

f.t <- fitCopula(tCopula(dim = d), U)
U.Rt <- cCopula(U, copula = f.t@copula) # transform with a fitted t copula
splom2(U.Rt, cex = 0.2) # still visible but not so clear

## Inverse (and check consistency)
U.N <- cCopula(U.RN, copula = fN @copula, inverse = TRUE)
```

```
U.t <- cCopula(U.Rt, copula = f.t@copula, inverse = TRUE)

tol <- 1e-14
stopifnot(
    all.equal(U, U.N),
    all.equal(U, U.t),
    all.equal(log(U.RN),
              cCopula(U, copula = fN @copula, log = TRUE), tolerance = tol),
    all.equal(log(U.Rt),
              cCopula(U, copula = f.t@copula, log = TRUE), tolerance = tol)
 )
```

---

cloud2-methods *Cloud Plot Methods ('cloud2') in Package 'copula'*

Description

Function and Methods cloud2() to draw (**lattice**) cloud plots of two-dimensional distributions from package **copula**.

Usage

```
## S4 method for signature 'matrix'
cloud2(x,
      xlim = range(x[,1], finite = TRUE),
      ylim = range(x[,2], finite = TRUE),
      zlim = range(x[,3], finite = TRUE),
      xlab = NULL, ylab = NULL, zlab = NULL,
      scales = list(arrows = FALSE, col = "black"),
      par.settings = standard.theme(color = FALSE), ...)
## S4 method for signature 'data.frame'
cloud2(x,
      xlim = range(x[,1], finite = TRUE),
      ylim = range(x[,2], finite = TRUE),
      zlim = range(x[,3], finite = TRUE),
      xlab = NULL, ylab = NULL, zlab = NULL,
      scales = list(arrows = FALSE, col = "black"),
      par.settings = standard.theme(color = FALSE), ...)
## S4 method for signature 'Copula'
cloud2(x, n,
      xlim = 0:1, ylim = 0:1, zlim = 0:1,
      xlab = quote(U[1]), ylab = quote(U[2]), zlab = quote(U[3]), ...)
## S4 method for signature 'mvdc'
cloud2(x, n,
      xlim = NULL, ylim = NULL, zlim = NULL,
      xlab = quote(X[1]), ylab = quote(X[2]), zlab = quote(X[3]), ...)
```

Arguments

- `x` a "matrix", "data.frame", "Copula" or a "mvdc" object.
- `xlim, ylim, zlim` the x-, y- and z-axis limits.
- `xlab, ylab, zlab` the x-, y- and z-axis labels.
- `scales` a `list` determining how the axes are drawn; see `cloud()`.
- `par.settings` see `cloud()`.
- `n` when `x` is not matrix-like: The sample size of the random sample drawn from `x`.
- `...` additional arguments passed to the underlying `cloud()`.

Value

An object of class "trellis" as returned by `cloud()`.

Methods

Cloud plots for objects of class "matrix" , "data.frame", "Copula" or "mvdc".

See Also

The **lattice**-based `splom2-methods` for data, and `wireframe2-methods` and `contourplot2-methods` for functions.

Examples

```
## For 'matrix' objects
cop <- gumbelCopula(2, dim = 3)
n <- 1000
set.seed(271)
U <- rCopula(n, copula = cop)
cloud2(U, xlab = quote(U[1]), ylab = quote(U[2]), zlab = quote(U[3]))

## For 'Copula' objects
set.seed(271)
cloud2(cop, n = n) # same as above

## For 'mvdc' objects
mvNN <- mvdc(cop, c("norm", "norm", "exp"),
             list(list(mean = 0, sd = 1), list(mean = 1), list(rate = 2)))
cloud2(mvNN, n = n)
```

coeffG *Coefficients of Polynomial used for Gumbel Copula*

Description

Compute the coefficients $a_{d,k}(\theta)$ involved in the generator (psi) derivatives and the copula density of Gumbel copulas.

For non-small dimensions d, these are numerically challenging to compute accurately.

Usage

```
coeffG(d, alpha,
       method = c("sort", "horner", "direct", "dsumSibuya",
                  paste("dsSib", eval(formals(dsumSibuya)$method), sep = ".")),
       log = FALSE, verbose = FALSE)
```

Arguments

- `d` number of coefficients, (the copula dimension), d >= 1.
- `alpha` parameter $1/\theta$ in $(0, 1]$; you may use `mpfr(alph, precBits = <n_prec>)` for higher precision methods (`"Rmpfr*"`) from package **Rmpfr**.
- `method` a `character` string, one of
  - `"sort"`: compute coefficients via $exp(log())$ pulling out the maximum, and sort.
  - `"horner"`: uses polynomial evaluation, our internal `polynEval()`.
  - `"direct"`: brute force approach.
  - `"dsSib.<FOO>"`: uses `dsumSibuya(..., method= "<FOO>")`.
- `log` logical determining if the logarithm (`log`) is to be returned.
- `verbose` logical indicating if some information should be shown, currently for `method == "sort"` only.

Value

a numeric vector of length `d`, of values

$$a_k(\theta, d) = (-1)^{d-k} \sum_{j=k}^{d} \alpha^j * s(d, j) * S(j, k), k \in \{1, \ldots, d\}.$$

Note

There are still known numerical problems (with non-"Rmpfr" methods; and those are slow), e.g., for d=100, alpha=0.8 and $sign(s(n, k)) = (-1)^{n-k}$.

As a consequence, the `method`s and its defaults may change in the future, and so the exact implementation of `coeffG()` is still considered somewhat experimental.

Examples

```
a.k  <- coeffG(16, 0.55)
plot(a.k, xlab = quote(k), ylab = quote(a[k]),
     main = "coeffG(16, 0.55)", log = "y", type = "o", col = 2)
a.kH <- coeffG(16, 0.55, method = "horner")
stopifnot(all.equal(a.k, a.kH, tol = 1e-11))# 1.10e-13 (64-bit Lnx, nb-mm4)
```

---

contour-methods *Methods for Contour Plots in Package 'copula'*

Description

Methods for function `contour` to draw contour lines aka a level plot for objects from package **copula**.

Usage

```
## S4 method for signature 'Copula'
contour(x, FUN,
                    n.grid = 26, delta = 0,
                    xlab = quote(u[1]), ylab = quote(u[2]),
                    box01 = TRUE, ...)
## S4 method for signature 'mvdc'
contour(x, FUN, xlim, ylim, n.grid = 26,
                    xlab = quote(x[1]), ylab = quote(x[2]),
   box01 = FALSE, ...)
```

Arguments

`x`	a "`Copula`" or a "`mvdc`" object.
`FUN`	the `function` to be plotted; typically `dCopula` or `pCopula`.
`n.grid`	the number of grid points used in each dimension. This can be a vector of length two, giving the number of grid points used in x- and y-direction, respectively; the function `FUN` will be evaluated on the corresponding (x,y)-grid.
`delta`	a small number in $[0, \frac{1}{2})$ influencing the evaluation boundaries. The x- and y-vectors will have the range `[0+delta, 1-delta]`, the default being `[0,1]`.
`xlab, ylab`	the x-axis and y-axis labels.
`xlim, ylim`	the `range` of the x and y variables, respectively.
`box01`	a logical specifying if a faint rectangle should be drawn on the boundary of $[0, 1]^2$ (often useful for copulas, but typically *not* for general multivariate distributions ("`mvdc`")).
`...`	further arguments for (the default method of) `contour()`, e.g., `nlevels`, `levels`, etc.

Methods

Contour lines are drawn for "`Copula`" or "`mvdc`" objects, see `x` in the *Arguments* section.

See Also

The `persp-methods` for "perspective" aka "3D" plots.

Examples

```
contour(frankCopula(-0.8), dCopula)
contour(frankCopula(-0.8), dCopula, delta=1e-6)
contour(frankCopula(-1.2), pCopula)
contour(claytonCopula(2), pCopula)

## the Gumbel copula density is "extreme"
## --> use fine grid (and enough levels):
r <- contour(gumbelCopula(3), dCopula, n=200, nlevels=100)
range(r$z)# [0, 125.912]
## Now superimpose contours of three resolutions:
contour(r, levels = seq(1, max(r$z), by=2), lwd=1.5)
contour(r, levels = (1:13)/2, add=TRUE, col=adjustcolor(1,3/4), lty=2)
contour(r, levels = (1:13)/4, add=TRUE, col=adjustcolor(2,1/2),
        lty=3, lwd=3/4)

x <- mvdc(gumbelCopula(3), c("norm", "norm"),
          list(list(mean = 0, sd =1), list(mean = 1)))
contour(x, dMvdc, xlim=c(-2, 2), ylim=c(-1, 3))
contour(x, pMvdc, xlim=c(-2, 2), ylim=c(-1, 3))
```

---

contourplot2-methods *Contour Plot Methods 'contourplot2' in Package 'copula'*

Description

Methods for `contourplot2()`, a version of `contourplot()` from **lattice**, to draw contour plots of two dimensional distributions from package **copula**.

Usage

```
## S4 method for signature 'matrix'
contourplot2(x, aspect = 1,
      xlim = range(x[,1], finite = TRUE),
      ylim = range(x[,2], finite = TRUE),
      xlab = NULL, ylab = NULL,
      cuts = 16, labels = !region,
      scales = list(alternating = c(1,1), tck = c(1,0)),
      region = TRUE, ...,
      col.regions = gray(seq(0.4, 1, length.out = 100)))
```

```
## S4 method for signature 'data.frame'
contourplot2(x, aspect = 1,
      xlim = range(x[,1], finite = TRUE),
      ylim = range(x[,2], finite = TRUE),
      xlab = NULL, ylab = NULL,
      cuts = 16, labels = !region,
      scales = list(alternating = c(1,1), tck = c(1,0)),
      region = TRUE, ...,
      col.regions = gray(seq(0.4, 1, length.out = 100)))
## S4 method for signature 'Copula'
contourplot2(x, FUN, n.grid = 26, delta = 0,
      xlim = 0:1, ylim = 0:1,
      xlab = quote(u[1]), ylab = quote(u[2]), ...)
## S4 method for signature 'mvdc'
contourplot2(x, FUN, n.grid = 26, xlim, ylim,
      xlab = quote(x[1]), ylab = quote(x[2]), ...)
```

Arguments

`x`	a "`matrix`", "`data.frame`", "`Copula`" or a "`mvdc`" object.
`aspect`	the aspect ratio.
`xlim, ylim`	the x- and y-axis limits.
`xlab, ylab`	the x- and y-axis labels. If at least one is `NULL`, useful `xlab` and `ylab` are determined automatically; the behavior depends on the class of `x`.
`cuts`	the number of levels; see `contourplot()`.
`labels`	a `logical` indicating whether the contour lines are labeled; see `contourplot()`.
`scales`	a `list` determining how the axes are drawn; see `contourplot()`.
`region`	a `logical` indicating whether regions between contour lines should be filled as in a level plot; see `contourplot()`.
`col.regions`	the colors of the regions if `region = TRUE`; see `contourplot()`.
`FUN`	the `function` to be plotted; typically `dCopula` or `pCopula`.
`n.grid`	the number of grid points used in each dimension. This can be a vector of length two, giving the number of grid points used in x- and y-direction, respectively; the function `FUN` will be evaluated on the corresponding (x,y)-grid.
`delta`	a small number in $[0, \frac{1}{2})$ influencing the evaluation boundaries. The x- and y-vectors will have the range `[0+delta, 1-delta]`, the default being `[0,1]`.
`...`	additional arguments passed to the underlying `contourplot()`.

Value

An object of class "trellis" as returned by `contourplot()`.

Methods

Contourplot plots for objects of class "`matrix`" , "`data.frame`", "`Copula`" or "`mvdc`".

See Also

The `contour-methods` for drawing perspective plots via base graphics.

The **lattice**-based `wireframe2-methods` for functions, and `cloud2-methods` and `splom2-methods` for data.

Examples

```
## For 'matrix' objects
## The Frechet--Hoeffding bounds W and M
n.grid <- 26
u <- seq(0, 1, length.out = n.grid)
grid <- expand.grid("u[1]" = u, "u[2]" = u)
W <- function(u) pmax(0, rowSums(u)-1) # lower bound W
M <- function(u) apply(u, 1, min) # upper bound M
x.W <- cbind(grid, "W(u[1],u[2])" = W(grid)) # evaluate W on 'grid'
x.M <- cbind(grid, "M(u[1],u[2])" = M(grid)) # evaluate M on 'grid'
contourplot2(x.W) # contour plot of W
contourplot2(x.M) # contour plot of M

## For 'Copula' objects
cop <- frankCopula(-4)
contourplot2(cop, pCopula) # the copula
contourplot2(cop, pCopula, xlab = "x", ylab = "y") # adjusting the labels
contourplot2(cop, dCopula) # the density

## For 'mvdc' objects
mvNN <- mvdc(gumbelCopula(3), c("norm", "norm"),
             list(list(mean = 0, sd = 1), list(mean = 1)))
xl <- c(-2, 2)
yl <- c(-1, 3)
contourplot2(mvNN, FUN = dMvdc, xlim = xl, ylim = yl, contour = FALSE)
contourplot2(mvNN, FUN = dMvdc, xlim = xl, ylim = yl)
contourplot2(mvNN, FUN = dMvdc, xlim = xl, ylim = yl, region = FALSE, labels = FALSE)
contourplot2(mvNN, FUN = dMvdc, xlim = xl, ylim = yl, region = FALSE)
contourplot2(mvNN, FUN = dMvdc, xlim = xl, ylim = yl,
             col.regions = colorRampPalette(c("royalblue3", "maroon3"), space="Lab"))
```

---

copFamilies — *Specific Archimedean Copula Families ("acopula" Objects)*

Description

Specific Archimedean families ("`acopula`" objects) implemented in the package **copula**.

These families are "classical" as from p. 116 of Nelsen (2007). More specifially, see Table 1 of Hofert (2011).

Usage

```
copAMH
copClayton
copFrank
copGumbel
copJoe
```

Details

All these are objects of the formal class "`acopula`".

`copAMH`: Archimedean family of Ali-Mikhail-Haq with parametric generator

$$\psi(t) = (1-\theta)/(\exp(t)-\theta),\ t \in [0,\infty],$$

with $\theta \in [0,1)$. The range of admissible Kendall's tau is [0,1/3). Note that the lower and upper tail-dependence coefficients are both zero, that is, this copula family does not allow for tail dependence.

`copClayton`: Archimedean family of Clayton with parametric generator

$$\psi(t) = (1+t)^{-1/\theta},\ t \in [0,\infty],$$

with $\theta \in (0,\infty)$. The range of admissible Kendall's tau, as well as that of the lower tail-dependence coefficient, is (0,1). Note that this copula does not allow for upper tail dependence.

`copFrank`: Archimedean family of Frank with parametric generator

$$-\log(1-(1-e^{-\theta})\exp(-t))/\theta,\ t \in [0,\infty]$$

with $\theta \in (0,\infty)$. The range of admissible Kendall's tau is (0,1). Note that this copula family does not allow for tail dependence.

`copGumbel`: Archimedean family of Gumbel with parametric generator

$$\exp(-t^{1/\theta}),\ t \in [0,\infty]$$

with $\theta \in [1,\infty)$. The range of admissible Kendall's tau, as well as that of the upper tail-dependence coefficient, is [0,1). Note that this copula does not allow for lower tail dependence.

`copJoe`: Archimedean family of Joe with parametric generator

$$1-(1-\exp(-t))^{1/\theta},\ t \in [0,\infty]$$

with $\theta \in [1,\infty)$. The range of admissible Kendall's tau, as well as that of the upper tail-dependence coefficient, is [0,1). Note that this copula does not allow for lower tail dependence.

Note that staying within one of these Archimedean families, all of them can be nested if two (generic) generator parameters θ_0, θ_1 satisfy $\theta_0 \le \theta_1$.

Value

A "`acopula`" object.

References

Nelsen, R. B. (2007). *An Introduction to Copulas* (2nd ed.). Springer.

Hofert, M. (2010). *Sampling Nested Archimedean Copulas with Applications to CDO Pricing*. Suedwestdeutscher Verlag fuer Hochschulschriften AG & Co. KG.

Hofert, M. (2011). Efficiently sampling nested Archimedean copulas. *Computational Statistics & Data Analysis* **55**, 57–70.

Hofert, M. and Mächler, M. (2011). Nested Archimedean Copulas Meet R: The nacopula Package. *Journal of Statistical Software* **39**(9), 1–20. http://www.jstatsoft.org/v39/i09/.

See Also

The class definition, "acopula". onacopula and setTheta for such Archimedean copulas with specific parameters.
getAcop accesses these families "programmatically".

Examples

```
## Print a copAMH object and its structure
copAMH
str(copAMH)

## Show admissible parameters for a Clayton copula
copClayton@paraInterval

## Generate random variates from a Log(p) distribution via V0 of Frank
p <- 1/2
copFrank@V0(100, -log(1-p))

## Plot the upper tail-dependence coefficient as a function in the
## parameter for Gumbel's family
curve(copGumbel@lambdaU(x), xlim = c(1, 10), ylim = c(0,1), col = 4)

## Plot Kendall's tau as a function in the parameter for Joe's family
curve(copJoe@tau(x), xlim = c(1, 10), ylim = c(0,1), col = 4)

## ------- Plot psi() and tau() - and properties of all families ----

## The copula families currently provided:
(famNms <- ls("package:copula", patt="^cop[A-Z]"))

op <- par(mfrow= c(length(famNms), 2),
          mar = .6+ c(2,1.4,1,1), mgp = c(1.1, 0.4, 0))
for(nm in famNms) { Cf <- get(nm)
   thet <- Cf@iTau(0.3)
   curve(Cf@psi(x, theta = thet), 0, 5,
         xlab = quote(x), ylab="", ylim=0:1, col = 2,
         main = substitute(list(NAM ~~~ psi(x, theta == TH), tau == 0.3),
                           list(NAM=Cf@name, TH=thet)))
   I <- Cf@paraInterval
   Iu <- pmin(10, I[2])
```

```
  curve(Cf@tau(x), I[1], Iu, col = 3,
        xlab = bquote(theta %in% .(format(I))), ylab = "",
        main = substitute(NAM ~~ tau(theta), list(NAM=Cf@name)))
}
par(op)

## Construct a bivariate Clayton copula with parameter theta
theta <- 2
C2 <- onacopula("Clayton", C(theta, 1:2))
C2@copula # is an "acopula" with specific parameter theta

curve(C2@copula@psi(x, C2@copula@theta),
      main = quote("Generator" ~~ psi ~~ " of Clayton A.copula"),
      xlab = quote(theta1), ylab = quote(psi(theta1)),
      xlim = c(0,5), ylim = c(0,1), col = 4)

## What is the corresponding Kendall's tau?
C2@copula@tau(theta) # 0.5

## What are the corresponding tail-dependence coefficients?
C2@copula@lambdaL(theta)
C2@copula@lambdaU(theta)

## Generate n pairs of random variates from this copula
U <- rnacopula(n = 1000, C2)
## and plot the generated pairs of random variates
plot(U, asp=1, main = "n = 1000 from  Clayton(theta = 2)")
```

Copula — *Density, Evaluation, and Random Number Generation for Copula Functions*

Description

Density (dCopula), distribution function (pCopula), and random generation (rCopula) for a copula object.

Usage

```
dCopula(u, copula, log=FALSE, ...)
pCopula(u, copula, ...)
rCopula(n, copula, ...)
```

Arguments

copula	an R object of class "Copula", (i.e., "copula" or "nacopula").
u	a vector of the copula dimension d or a matrix with d columns, giving the points where the density or distribution function needs to be evaluated. Note that in all cases, values outside of the cube $[0,1]^d$ are treated equivalently to those on the cube boundary. So, e.g., the density is zero.

log	logical indicating if the $\log(f(\cdot))$ should be returned instead of $f(\cdot)$.
n	(for rCopula():) number of observations to be generated.
...	further optional arguments for some methods, e.g., method.

Details

The density (dCopula) and distribution function (pCopula) methods for Archimedean copulas now use the corresponding function slots of the Archimedean copula objects, such as copClayton, copGumbel, etc.

The distribution function of a t copula uses pmvt from package **mvtnorm**; similarly, the density (dCopula) calls dmvt from **mvtnorm**. The normalCopula methods use dmvnorm and pmvnorm from the same package.

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

Value

dCopula() gives the density, pCopula() gives the distribution function, and rCopula() generates random variates.

References

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

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

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

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

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

See Also

the copula and acopula classes, the acopula families, acopula-families. Constructor functions such as ellipCopula, archmCopula, fgmCopula.

Examples

```
norm.cop <- normalCopula(0.5)
norm.cop
## one d-vector =^= 1-row matrix, works too :
dCopula(c(0.5, 0.5), norm.cop)
pCopula(c(0.5, 0.5), norm.cop)

u <- rCopula(100, norm.cop)
plot(u)
dCopula(u, norm.cop)
pCopula(u, norm.cop)
```

```
persp  (norm.cop, dCopula)
contour(norm.cop, pCopula)

## a 3-dimensional normal copula
u <- rCopula(1000, normalCopula(0.5, dim = 3))
if(require(scatterplot3d))
  scatterplot3d(u)

## a 3-dimensional clayton copula
cl3 <- claytonCopula(2, dim = 3)
v <- rCopula(1000, cl3)
pairs(v)
if(require(scatterplot3d))
  scatterplot3d(v)

## Compare with the "nacopula" version :
fu1 <- dCopula(v, cl3)
fu2 <- copClayton@dacopula(v, theta = 2)
Fu1 <- pCopula(v, cl3)
Fu2 <- pCopula(v, onacopula("Clayton", C(2.0, 1:3)))
## The density and cumulative values are the same:
stopifnot(all.equal(fu1, fu2, tolerance= 1e-14),
          all.equal(Fu1, Fu2, tolerance= 1e-15))
```

`copula-class` *Mother Classes "Copula", etc of all Copulas in the Package*

Description

A copula is a multivariate distribution with uniform margins. The virtual class `"Copula"` is the mother (or “super class”) of all copula classes in the package **copula** which encompasses classes of the former packages **nacopula** and **copula**.

The virtual class `"parCopula"` extends `"Copula"` and is the super class of all copulas that can be fully *par*ametrized and hence fitted to data. For these, at least the `dim()` method must be well defined.

The virtual class `"copula"` extends `"parCopula"` and is the mother of all copula classes from former package **copula**. It has set of slots for the dimension and parameter vector, see below.

The virtual class `"xcopula"` extends `"parCopula"` and contains a slot `copula`; an (“actual”) class example are the rotated copulas, `rotCopula`.

Objects from the Class

Objects are typically created by are by `tCopula()`, `evCopula()`, etc.

Note that the virtual class `"Copula"`, is simply the union (see `setClassUnion`) of the two classes `"copula"` and `"nacopula"`.

Slots

Class "copula" (and all its subclasses) have slots

dimension: an "integer" (of length 1), the copula dimension d.

parameters: "numeric" vector of parameter values, can be NA (i.e., NA_real_).

param.names: "character" vector of parameter names (and hence of the same length as parameters).

param.lowbnd: lower bounds for the parameters, of class "numeric".

param.upbnd: upper bounds for the parameters, of class "numeric".

fullname: **deprecated**; 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 such copulas can be created via 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

Help for the (sub)classes archmCopula, ellipCopula, evCopula, and fgmCopula.

The Archimedean and nested Archimedean classes (from former package **nacopula**), with a more extensive list of slots (partly instead of methods), acopula, and nacopula.

Examples

```
hc <- evCopula("husler", 1.25)
dim(hc)
smoothScatter(u <- rCopula(2^11, hc))
lambda   (hc)
tau (hc)
rho(hc)
str(hc)
```

corKendall *(Fast) Computation of Pairwise Kendall's Taus*

Description

For a data matrix x, compute the Kendall's tau "correlation" matrix, i.e., all pairwise Kendall's taus between the columns of x.

By default and when x has no missing values (NAs), the fast $O(nlog(n))$ algorithm of cor.fk() is used.

Usage

```
corKendall(x, checkNA = TRUE,
           use = if(checkNA && anyNA(x)) "pairwise" else "everything")
```

Arguments

x	data, a n x p matrix (or less efficiently a data.frame), or a numeric vector which is treated as n x 1 matrix.
checkNA	logical indicating if x should be checked for NAs and in the case of NA's *and* when use is not specified (missing), cor(*, use = "pairwise") should be used. Note that corKendall(x, checkNA = FALSE) will produce an error when x has NA's.
use	a string to determine the treatment of NAs in x, see cor; its default determined via checkNA. When this differs from "everything", R's cor is used; otherwise **pcaPP**'s cor.fk() which cannot deal with NAs.

Value

The $p \times p$ matrix K of pairwise Kendall's taus, with K[i,j] := tau(x[,i], x[,j]).

See Also

cor.fk() from **pcaPP** (used by default when there are no missing values (NAs) in x).

etau() or fitCopula(*, method = "itau") make use of corKendall().

Examples

```
## If there are no NA's, corKendall() is faster than cor(*, "kendall")
## and gives the same :

system.time(C1 <- cor(swiss, method="kendall"))
system.time(C2 <- corKendall(swiss))
stopifnot(all.equal(C1, C2,  tol = 1e-5))

## In the case of missing values (NA), corKendall() reverts to
## cor(*, "kendall", use = "pairwise") {no longer very fast} :
```

```
swM <- swiss # shorter names and three missings:
colnames(swM) <- abbreviate(colnames(swiss), min=6)
swM[1,2] <- swM[7,3] <- swM[25,5] <- NA
(C3 <- corKendall(swM)) # now automatically uses the same as
stopifnot(identical(C3, cor(swM, method="kendall", use="pairwise")))
## and is quite close to the non-missing "truth":
stopifnot(all.equal(unname(C3), unname(C2), tol = 0.06)) # rel.diff.= 0.055

try(corKendall(swM, checkNA=FALSE)) # --> Error
## the error is really from  pcaPP::cor.fk(swM)
```

---

dDiag *Density of the Diagonal of (Nested) Archimedean Copulas*

Description

Evaluate the density of the diagonal of a d-dimensional (nested) Archimedean copula. Note that the diagonal of a copula is a cumulative distribution function. Currently, only Archimedean copulas are implemented.

Usage

```
dDiag(u, cop, log=FALSE)
```

Arguments

u	a numeric vector of evaluation points.
cop	a (nested) Archimedean copula object of class "outer_nacopula". This also determines the dimension via the comp slot
log	logical indicating if the log of the density of the diagonal should be returned instead of just the diagonal density.

Value

A numeric vector containing the values of the density of the diagonal of the Archimedean copula at u.

References

Hofert, M., Mächler, M., and McNeil, A. J. (2013). Archimedean Copulas in High Dimensions: Estimators and Numerical Challenges Motivated by Financial Applications. *Journal de la Société Française de Statistique* **154**(1), 25–63.

See Also

acopula class, dnacopula.

Examples

```
th. <- c(0.1, 0.2, 0.5, 0.8, 1.4, 2., 5.)
curve(dDiag(x, cop=onacopulaL("Clayton", list(th.[1], 1:3))), 0, 1,
      n=1000, ylab="dDiag(x, *)", main="Diagonal densities of Clayton")
abline(h=0, lty=3)
for(j in 2:length(th.))
  curve(dDiag(x, cop=onacopulaL("Clayton", list(th.[j], 1:3))), add=TRUE,
     col=j, n=1000)
legend("topleft", do.call(expression, lapply(th., function(th)
                                 substitute(theta == TH, list(TH=th)))),
       lty = 1, col=seq_along(th.), bty="n")
```

---

describeCop *Copula (Short) Description as String*

Description

Describe a copula object, i.e., its basic properties as a string. This is a utility used when `print()`ing or `plot()`ting copulas, e.g., after a fitting.

Usage

```
describeCop(x, kind = c("short", "very short", "long"), prefix = "", ...)
```

Arguments

`x`	a `copula` object, or a generalization such as `parCopula`.
`kind`	a `character` string specifying the size (or "complexity" of the copula description desired.
`prefix`	a string to be prefixed to the returned string, which can be useful for indentation in describing extended copulas such as Khoudraji copulas.
`...`	further arguments; unused currently.

Value

a `character` string.

Methods

`signature(x = "archmCopula", kind = "ANY")` ..

`signature(x = "copula", kind = "character")` ..

`signature(x = "copula", kind = "missing")` ..

`signature(x = "ellipCopula", kind = "character")` ..

`signature(x = "fgmCopula", kind = "ANY")` ..

`signature(x = "xcopula", kind = "ANY")` ..

See Also

Copula class definition copula;

Examples

```
## FIXME
```

---

dnacopula *Density Evaluation for (Nested) Archimedean Copulas*

Description

For a (nested) Archimedean copula (object of class nacopula) x, dCopula(u, x) (or also currently still dnacopula(x, u)) evaluates the density of x at the given vector or matrix u.

Usage

```
## S4 method for signature 'matrix,nacopula'
dCopula(u, copula, log=FALSE, ...)

## *Deprecated*:
dnacopula(x, u, log=FALSE, ...)
```

Arguments

copula, x	an object of class "outer_nacopula".
u	argument of the copula x. Note that u can be a matrix in which case the density is computed for each row of the matrix and the vector of values is returned.
log	logical indicating if the log of the density should be returned.
...	optional arguments passed to the copula's dacopula function (slot), such as n.MC (non-negative integer) for possible Monte Carlo evaluation (see dacopula in acopula).

Details

If it exists, the density of an Archimedean copula C with generator ψ at $\boldsymbol{u} \in (0,1)^d$ is given by

$$c(\boldsymbol{u}) = \psi^{(d)}(\psi^{-1}(u_1) + \ldots + \psi^{-1}(u_d)) \prod_{j=1}^{d} (\psi^{-1}(u_j))' = \frac{\psi^{(d)}(\psi^{-1}(u_1) + \ldots + \psi^{-1}(u_d))}{\prod_{j=1}^{d} \psi'(\psi^{-1}(u_j))}.$$

Value

A numeric vector containing the values of the density of the Archimedean copula at u.

Note

dCopula(u, copula) is a *generic* function with methods for *all* our copula classes, see dCopula.

References

Hofert, M., Mächler, M., and McNeil, A. J. (2012). Likelihood inference for Archimedean copulas in high dimensions under known margins. *Journal of Multivariate Analysis* **110**, 133–150.

Hofert, M., Mächler, M., and McNeil, A. J. (2013). Archimedean Copulas in High Dimensions: Estimators and Numerical Challenges Motivated by Financial Applications. *Journal de la Société Française de Statistique* **154**(1), 25–63.

See Also

For more details about the derivatives of an Archimedean generator, see, for example, `absdPsi` in class `acopula`.

Examples

```
## Construct a twenty-dimensional Gumbel copula with parameter chosen
## such that Kendall's tau of the bivariate margins is 0.25.
theta <- copJoe@iTau(.25)
C20 <- onacopula("J", C(theta, 1:20))

## Evaluate the copula density at the point u = (0.5,...,0.5)
u <- rep(0.5, 20)
dCopula(u, C20)

## the same with Monte Carlo based on 10000 simulated "frailties"
dCopula(u, C20, n.MC = 10000)

## Evaluate the exact log-density at several points
u <- matrix(runif(100), ncol=20)
dCopula(u, C20, log = TRUE)

## Back-compatibility check
stopifnot(identical( dCopula (u, C20), suppressWarnings(
                    dnacopula(C20, u))),
          identical( dCopula (u, C20, log = TRUE), suppressWarnings(
                    dnacopula(C20, u, log = TRUE))))
```

`ellipCopula` *Construction of Elliptical Copula Class Objects*

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, df.min = 0.01)
```

Arguments

`family`	a character string specifying the family of an elliptical copula. Must be `"normal"` (the default) or `"t"`.
`param`	a numeric vector specifying the parameter values; `P2p()` accesses this vector, whereas `p2P()` and `getSigma()` provide the corresponding "P" matrix, see below.
`dim`	the dimension of the copula.
`dispstr`	a character string specifying the type of the symmetric positive definite matrix characterizing the elliptical copula. Currently available structures are `"ex"` for **ex**changeable, `"ar1"` for $AR(1)$, `"toep"` for Toeplitz (`toeplitz`), and `"un"` for **un**structured.
`df`	integer value specifying the number of degrees of freedom of the multivariate t distribution used to construct the t copulas.
`df.fixed`	logical specifying if the degrees of freedom `df` will be considered as a parameter (to be estimated) or not. The default, `FALSE`, means that `df` is to be estimated if the object is passed as argument to `fitCopula`.
`df.min`	non-negative number; the strict lower bound for `df`, mainly during fitting when `df.fixed=FALSE`, with `fitCopula`.
`...`	currently nothing.

Value

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

Note

`ellipCopula()` is a wrapper for `normalCopula()` and `tCopula()`.

The `pCopula()` methods for the normal- and t-copulas accept optional arguments to be passed to the underlying (numerical integration) algorithms from package **mvtnorm**'s `pmvnorm` and `pmvt`, respectively, notably `algorithm`, see `GenzBretz`, or `abseps` which defaults to `0.001`. ## For smaller copula dimension 'd', alternatives are available and ## non-random, see ?GenzBretz from package 'mvtnorm'

See Also

`p2P()`, and `getSigma()` for construction and extraction of the dispersion matrix P or $Sigma$ matrix of (generalized) correlations.

`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)
getSigma(t.cop) # P matrix (with diagonal = 1)
```

```
## dispersion "AR1" :
nC.7 <- normalCopula(0.8, dim = 7, dispstr = "ar1")
getSigma(nC.7) ##-> toeplitz( (1  0.8  0.8^2  0.8^3  ... 0.8^6) ) matrix

## from the wrapper
norm.cop <- ellipCopula("normal", param = c(0.5, 0.6, 0.7),
                        dim = 3, dispstr = "un")
if(require("scatterplot3d") && dev.interactive(orNone=TRUE)) {
  ## 3d scatter plot of 1000 random observations
  scatterplot3d(rCopula(1000, norm.cop))
  scatterplot3d(rCopula(1000, t.cop))
}
set.seed(12)
uN <- rCopula(512, norm.cop)
set.seed(2); pN1 <- pCopula(uN, norm.cop)
set.seed(3); pN2 <- pCopula(uN, norm.cop)
stopifnot(all.equal(pN1, pN2, 1e-4))# see 5.711e-5
(Xtras <- copula:::doExtras())
if(Xtras) { ## a bit more accurately:
  set.seed(4); pN1. <- pCopula(uN, norm.cop, abseps = 1e-9)
  set.seed(5); pN2. <- pCopula(uN, norm.cop, abseps = 1e-9)
  stopifnot(all.equal(pN1., pN2., 1e-5))# see 3.397e-6
  ## but increasing the required precision (e.g., abseps=1e-15) does *NOT* help
}

## For smaller copula dimension 'd', alternatives are available and
## non-random, see ?GenzBretz from package 'mvtnorm' :
require("mvtnorm")# -> GenzBretz(), Miva(), and TVPACK() are available
## Note that Miwa() would become very slow for dimensions 5, 6, ..
set.seed(4); pN1.M <- pCopula(uN, norm.cop, algorithm = Miwa(steps = 512))
set.seed(5); pN2.M <- pCopula(uN, norm.cop, algorithm = Miwa(steps = 512))
stopifnot(all.equal(pN1.M, pN2.M, tol= 1e-15))# *no* randomness
set.seed(4); pN1.T <- pCopula(uN, norm.cop, algorithm = TVPACK(abseps = 1e-10))
set.seed(5); pN2.T <- pCopula(uN, norm.cop, algorithm = TVPACK(abseps = 1e-14))
stopifnot(all.equal(pN1.T, pN2.T, tol= 1e-15))# *no* randomness (but no effect of 'abseps')


## Versions with unspecified parameters:
tCopula()
allEQ <- function(u,v) all.equal(u, v, tolerance=0)
stopifnot(allEQ(ellipCopula("norm"), normalCopula()),
          allEQ(ellipCopula("t"), tCopula()))
tCopula(dim=3)
tCopula(dim=4, df.fixed=TRUE)
tCopula(dim=5, disp = "toep", df.fixed=TRUE)
normalCopula(dim=4, disp = "un")
```

---

ellipCopula-class *Class "ellipCopula" of Elliptical Copulas*

Description

Copulas generated from elliptical multivariate distributions, notably Normal- and t-copulas (of specific class "normalCopula" or "tCopula", respectively).

Objects from the Class

Objects are typically created by ellipCopula(), normalCopula(), or tCopula().

Slots

dispstr: "character" string indicating how the dispersion matrix is parameterized; one of "ex", "ar1", "toep", or "un", see the dispstr argument of ellipCopula().

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

parameters: a numeric, (vector of) the parameter value(s).

param.names: character vector with names for the parameters slot, of the same length.

param.lowbnd: numeric vector of lower bounds for the parameters slot, of the same length.

param.upbnd: upper bounds for parameters, analogous to parm.lowbnd.

fullname: **deprecated**; object of class "character", family names of the copula.

Extends

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

Methods

Many methods are available, notably dCopula, pCopula, and rCopula. Use, e.g., methods(class = "tCopula") to find others.

See Also

ellipCopula which also documents tCopula() and normalCopula(); copula-class.

---

emde *Minimum Distance Estimators for (Nested) Archimedean Copulas*

Description

Compute minimum distance estimators for (nested) Archimedean copulas.

Usage

```
emde(u, cop,
     method=c("mde.chisq.CvM", "mde.chisq.KS",
              "mde.gamma.CvM", "mde.gamma.KS"),
     interval=initOpt(cop@copula@name),
     include.K = FALSE, repara = TRUE, ...)
```

Arguments

`u` $n \times d$-matrix of (pseudo-)observations (each value in $[0, 1]$) from the copula, where n denotes the sample size and d the dimension.

`cop` `outer_nacopula` to be estimated (currently only Archimedean copulas are provided).

`method` a `character` string specifying the distance method, which has to be one (or a unique abbreviation) of

- `"mde.chisq.CvM"` map to an Erlang distribution and using a chi-square distribution and Cramér-von Mises distance;
- `"mde.chisq.KS"` map to an Erlang distribution and using a chi-square distribution and Kolmogorov-Smirnov distance;
- `"mde.gamma.CvM"` map to an Erlang distribution and using a Erlang distribution and Cramér-von Mises distance;
- `"mde.gamma.KS"` map to an Erlang distribution and using a Kolmogorov-Smirnov distance.

The four methods are described in Hofert et al. (2013); see also the 'Details' section.

`interval` bivariate vector denoting the interval where optimization takes place. The default is computed as described in Hofert et al. (2013).

`include.K` logical indicating whether the last component, the (possibly numerically challenging) Kendall distribution function K, is used (`include.K=TRUE`) or not. Note that the default is `FALSE` here, where it is `TRUE` in the underlying `htrafo()` function.

`repara` logical indicating whether the distance function to be optimized is reparametrized (the default); see the code for more details.

`...` additional arguments passed to `optimize()`.

Details

First, `htrafo` is applied to map the $n \times d$-matrix of given realizations to a $n \times d$-matrix or $n \times (d-1)$-matrix, depending on whether the last component is included (`include.K=TRUE`) or not. Second, using either the sum of squares of the standard normal quantile function (`method="mde.chisq.CvM"` and `method="mde.chisq.KS"`) or the sum of negative logarithms (`method="mde.gamma.CvM"` and `method="mde.gamma.KS"`), a map to a chi-square or an Erlang distribution is applied, respectively. Finally, a Cramér-von Mises (`method="mde.chisq.CvM"` and `method="mde.gamma.CvM"`) or Kolmogorov-Smirnov (`method="mde.chisq.KS"` and `method="mde.gamma.KS"`) distance is applied. This is repeated in an optimization until the copula parameter is found such that this distance is minimized.

Note that the same transformations as described above are applied for goodness-of-fit testing; see the 'See Also' section).

Value

`list` as returned by `optimize`, including the minimum distance estimator.

References

Hofert, M., Mächler, M., and McNeil, A. J. (2013). Archimedean Copulas in High Dimensions: Estimators and Numerical Challenges Motivated by Financial Applications. *Journal de la Société Française de Statistique* **154**(1), 25–63.

Hering, C. and Hofert, M. (2014), Goodness-of-fit tests for Archimedean copulas in high dimensions, *Innovations in Quantitative Risk Management*.

See Also

enacopula (wrapper for different estimators), gofCopula (wrapper for different goodness-of-fit tests), htrafo (transformation to a multivariate uniform distribution), and K (Kendall distribution function).

Examples

```
tau <- 0.25
(theta <- copGumbel@iTau(tau)) # 4/3
d <- 20
(cop <- onacopulaL("Gumbel", list(theta,1:d)))

set.seed(1)
n <- 200
U <- rnacopula(n, cop)

(meths <- eval(formals(emde)$method)) # "mde.chisq.CvM", ...
fun <- function(meth, u, cop, theta){
run.time <- system.time(val <- emde(u, cop=cop, method=meth)$minimum)
list(value=val, error=val-theta, utime.ms=1000*run.time[[1]])
}
(res <- sapply(meths, fun, u=U, cop=cop, theta=theta))
```

---

emle *Maximum Likelihood Estimators for (Nested) Archimedean Copulas*

Description

Compute (simulated) maximum likelihood estimators for (nested) Archimedean copulas.

Usage

```
emle(u, cop, n.MC=0, optimizer="optimize", method,
     interval=initOpt(cop@copula@name),
     start=list(theta=initOpt(cop@copula@name, interval=FALSE, u=u)),
     ...)
.emle(u, cop, n.MC=0,
      interval=initOpt(cop@copula@name), ...)
```

Arguments

u	$n \times d$-matrix of (pseudo-)observations (each value in $[0, 1]$) from the copula, with n the sample size and d the dimension.
cop	outer_nacopula to be estimated (currently only non-nested, that is, Archimedean copulas are admitted).
n.MC	integer, if positive, *simulated* maximum likelihood estimation (SMLE) is used with sample size equal to n.MC; otherwise (n.MC=0), MLE. In SMLE, the dth generator derivative and thus the copula density is evaluated via (Monte Carlo) simulation, whereas MLE uses the explicit formulas for the generator derivatives; see the details below.
optimizer	a string or NULL, indicating the optimizer to be used, where NULL means to use optim via the standard R function mle() from package **stats4**, whereas the default, "optimize" uses optimize via the R function mle2() from package **bbmle**.
method	only when optimizer is NULL or "optim", the method to be used for optim.
interval	bivariate vector denoting the interval where optimization takes place. The default is computed as described in Hofert et al. (2012).
start	list of initial values, passed through.
...	additional parameters passed to optimize.

Details

Exact formulas for the generator derivatives were derived in Hofert et al. (2012). Based on these formulas one can compute the (log-)densities of the Archimedean copulas. Note that for some densities, the formulas are numerically highly non-trivial to compute and considerable efforts were put in to make the computations numerically feasible even in large dimensions (see the source code of the Gumbel copula, for example). Both MLE and SMLE showed good performance in the simulation study conducted by Hofert et al. (2013) including the challenging 100-dimensional case. Alternative estimators (see also enacopula) often used because of their numerical feasibility, might break down in much smaller dimensions.

Note: SMLE for Clayton currently faces serious numerical issues and is due to further research. This is only interesting from a theoretical point of view, since the exact derivatives are known and numerically non-critical to evaluate.

Value

emle an R object of class "mle2" (and thus useful for obtaining confidence intervals) with the (simulated) maximum likelihood estimator.

.emle list as returned by optimize() including the maximum likelihood estimator (does not confidence intervals but is typically faster).

References

Hofert, M., Mächler, M., and McNeil, A. J. (2012). Likelihood inference for Archimedean copulas in high dimensions under known margins. *Journal of Multivariate Analysis* **110**, 133–150.

Hofert, M., Mächler, M., and McNeil, A. J. (2013). Archimedean Copulas in High Dimensions: Estimators and Numerical Challenges Motivated by Financial Applications. *Journal de la Société Française de Statistique* **154**(1), 25–63.

See Also

`mle2` from package **bbmle** and `mle` from **stats4** on which `mle2` is modeled. `enacopula` (wrapper for different estimators). `demo(opC-demo)` and `demo(GIG-demo)` for examples of two-parameter families.

Examples

```
tau <- 0.25
(theta <- copGumbel@iTau(tau)) # 4/3
d <-  20
(cop <- onacopulaL("Gumbel", list(theta,1:d)))

set.seed(1)
n <- 200
U <- rnacopula(n,cop)

## Estimation
system.time(efm <- emle(U, cop))
summary(efm) # using bblme's 'mle2' method

## Profile likelihood plot [using S4 methods from bbmle/stats4] :
pfm <- profile(efm)
ci  <- confint(pfm, level=0.95)
ci
stopifnot(ci[1] <= theta, theta <= ci[2])
plot(pfm)               # |z| against theta, |z| = sqrt(deviance)
plot(pfm, absVal=FALSE, #  z  against theta
     show.points=TRUE) # showing how it's interpolated
## and show the true theta:
abline(v=theta, col="lightgray", lwd=2, lty=2)
axis(1, pos = 0, at=theta, label=quote(theta[0]))

## Plot of the log-likelihood, MLE  and  conf.int.:
logL <- function(x) -efm@minuslogl(x)
       # == -sum(copGumbel@dacopula(U, theta=x, log=TRUE))
logL. <- Vectorize(logL)
I <- c(cop@copula@iTau(0.1), cop@copula@iTau(0.4))
curve(logL., from=I[1], to=I[2], xlab=quote(theta),
      ylab="log-likelihood",
      main="log-likelihood for Gumbel")
abline(v = c(theta, efm@coef), col="magenta", lwd=2, lty=2)
axis(1, at=c(theta, efm@coef), padj = c(-0.5, -0.8), hadj = -0.2,
     col.axis="magenta", label= expression(theta[0], hat(theta)[n]))
abline(v=ci, col="gray30", lwd=2, lty=3)
text(ci[2], extendrange(par("usr")[3:4], f= -.04)[1],
     "95% conf. int.", col="gray30", adj = -0.1)
```

enacopula *Estimation Procedures for (Nested) Archimedean Copulas*

Description

A set of ten different estimators, currently for one-parameter Archimedean copulas, of possibly quite high dimensions.

Usage

```
enacopula(u, cop,
          method = c("mle", "smle", "dmle",
                     "mde.chisq.CvM", "mde.chisq.KS",
                     "mde.gamma.CvM", "mde.gamma.KS",
                     "tau.tau.mean", "tau.theta.mean", "beta"),
          n.MC = if (method == "smle") 10000 else 0,
          interval = initOpt(cop@copula@name),
          xargs = list(), ...)
```

Arguments

u
: $n \times d$-matrix of (pseudo-)observations (each value in $[0, 1]$) from the copula to be estimated, where n denotes the sample size and d the dimension. Consider applying the function `pobs` first in order to obtain `u`.

cop
: `outer_nacopula` to be estimated (currently only Archimedean copulas are provided).

method
: a `character` string specifying the estimation method to be used, which has to be one (or a unique abbreviation) of

  - `"mle"` maximum likelihood estimator (MLE) computed via `.emle`.
  - `"smle"` simulated maximum likelihood estimator (SMLE) computed with the function `.emle`, where `n.MC` gives the Monte Carlo sample size.
  - `"dmle"` MLE based on the diagonal (DMLE); see `edmle`.
  - `"mde.chisq.CvM"` minimum distance estimator based on the chisq distribution and Cramér-von Mises distance; see `emde`.
  - `"mde.chisq.KS"` minimum distance estimation based on the chisq distribution and Kolmogorov-Smirnov distance; see `emde`.
  - `"mde.gamma.CvM"` minimum distance estimation based on the Erlang distribution and Cramér-von Mises distance; see `emde`.
  - `"mde.gamma.KS"` minimum distance estimation based on the Erlang distribution and Kolmogorov-Smirnov distance; see `emde`.
  - `"tau.tau.mean"` averaged pairwise Kendall's tau estimator
  - `"tau.theta.mean"` average of pairwise Kendall's tau estimators
  - `"beta"` multivariate Blomqvist's beta estimator

n.MC
: only for `method = "smle"`: `integer`, sample size for simulated maximum likelihood estimation.

`interval` bivariate vector denoting the interval where optimization takes place. The default is computed as described in Hofert et al. (2012). Used for all methods except "tau.tau.mean" and "tau.theta.mean".

`xargs` list of additional arguments for the chosen estimation method.

`...` additional arguments passed to `optimize`.

Details

`enacopula` serves as a wrapper for the different implemented estimators and provides a uniform framework to utilize them. For more information, see the single estimators as given in the section 'See Also'.

Note that Hofert, Mächler, and McNeil (2013) compared these estimators. Their findings include a rather poor performance and numerically challenging problems of some of these estimators. In particular, the estimators obtained by `method="mde.gamma.CvM"`, `method="mde.gamma.KS"`, `method="tau.theta.mean"`, and `method="beta"` should be used with care (or not at all). Overall, MLE performed best (by far).

Value

the estimated parameter, $\hat{\theta}$, that is, currently a number as only one-parameter Archimedean copulas are considered.

References

Hofert, M., Mächler, M., and McNeil, A. J. (2012). Likelihood inference for Archimedean copulas in high dimensions under known margins. *Journal of Multivariate Analysis* **110**, 133–150.

Hofert, M., Mächler, M., and McNeil, A. J. (2013). Archimedean Copulas in High Dimensions: Estimators and Numerical Challenges Motivated by Financial Applications. *Journal de la Société Française de Statistique* **154**(1), 25–63.

See Also

`emle` which returns an object of "`mle`" providing useful methods not available for other estimators. `demo(opC-demo)` and `vignette("GIG", package="copula")` for examples of two-parameter families. `edmle` for the diagonal maximum likelihood estimator. `emde` for the minimum distance estimators. `etau` for the estimators based on Kendall's tau. `ebeta` for the estimator based on Blomqvist's beta.

Examples

```
tau <- 0.25
(theta <- copGumbel@iTau(tau)) # 4/3
d <- 12
(cop <- onacopulaL("Gumbel", list(theta,1:d)))

set.seed(1)
n <- 100
U <- rnacopula(n, cop)
```

```
meths <- eval(formals(enacopula)$method)

fun <- function(meth, u, cop, theta) {
run.time <- system.time(val <- enacopula(u, cop=cop, method=meth))
list(value=val, error=val-theta, utime.ms=1000*run.time[[1]])
}
t(res <- sapply(meths, fun, u=U, cop=cop, theta=theta))
```

---

estim.misc *Various Estimators for (Nested) Archimedean Copulas*

Description

Various Estimators for (Nested) Archimedean Copulas, namely,

ebeta Method-of-moments-like estimator based on (a multivariate version of) Blomqvist'sbeta.

edmle Maximum likelihood estimator based on the diagonal of a (nested) Archimedean copula.

etau Method-of-moments-like estimators based on (bivariate) Kendall's tau.

Usage

```
ebeta(u, cop, interval = initOpt(cop@copula@name), ...)
edmle(u, cop, interval = initOpt(cop@copula@name), warn=TRUE, ...)
 etau(u, cop, method = c("tau.mean", "theta.mean"), warn=TRUE, ...)
```

Arguments

`u`	$n \times d$-matrix of (pseudo-)observations (each value in $[0, 1]$) from the copula, where n denotes the sample size and d the dimension.
`cop`	`outer_nacopula` to be estimated (currently only Archimedean copulas are provided).
`interval`	bivariate vector denoting the interval where optimization takes place. The default is computed as described in Hofert et al. (2013).
`method`	a character string specifying the method (only for `etau`), which has to be one (or a unique abbreviation) of `"tau.mean"` method-of-moments-like estimator based on the average of pairwise sample versions of Kendall's tau; `"theta.mean"` average of the method-of-moments-like Kendall's tau estimators.
`warn`	logical indicating if warnings are printed: `edmle()` for the family of `"Gumbel"` if the diagonal maximum-likelihood estimator is smaller than 1. `etau()` for the family of `"AMH"` if tau is outside $[0, 1/3]$ and in general if at least one of the computed pairwise sample versions of Kendall's tau is negative.
`...`	additional arguments passed to `corKendall` (for `etau`, but see 'Details'), to `optimize` (for `edmle`), or to `safeUroot` (for `ebeta`).

Details

For ebeta, the parameter is estimated with a method-of-moments-like procedure such that the population version of the multivariate Blomqvist's beta matches its sample version.

Note that the copula diagonal is a distribution function and the maximum of all components of a random vector following the copula is distributed according to this distribution function. For edmle, the parameter is estimated via maximum-likelihood estimation based on the diagonal.

For etau, corKendall(u, ...) is used and if there are no NAs in u, by default (if no additional arguments are provided), corKendall() calls the $O(nlog(n))$ fast cor.fk() from package **pcaPP** instead of the $O(n^2)$ cor(*, method="kendall"). Conversely, when u has NAs, by default, corKendall(u, ...) will use cor(u, method="kendall", use = "pairwise") such that etau(u, *) will work.
Furthermore, method="tau.mean" means that the average of sample versions of Kendall's tau are computed first and then the parameter is determined such that the population version of Kendall's tau matches this average (if possible); the method="theta.mean" stands for first computing all pairwise Kendall's tau estimators and then returning the mean of these estimators.

For more details, see Hofert et al. (2013).

Note that these estimators should be used with care; see the performance results in Hofert et al. (2013). In particular, etau should be used with the (default) method "tau.mean" since "theta.mean" is both slower and more prone to errors.

Value

ebeta the return value of safeUroot (that is, typically almost the same as the value of uniroot) giving the Blomqvist beta estimator.

edmle list as returned by optimize, including the diagonal maximum likelihood estimator.

etau method-of-moments-like estimator based on Kendall's tau for the chosen method.

References

Hofert, M., Mächler, M., and McNeil, A. J. (2013). Archimedean Copulas in High Dimensions: Estimators and Numerical Challenges Motivated by Financial Applications. *Journal de la Société Française de Statistique* **154**(1), 25–63.

See Also

corKendall().

The more sophisticated estimators emle (Maximum Likelihood) and emde (Minimum Distance). enacopula (wrapper for different estimators).

Examples

```
tau <- 0.25
(theta <- copGumbel@iTau(tau)) # 4/3 = 1.333..
d <- 20
(cop <- onacopulaL("Gumbel", list(theta,1:d)))

set.seed(1)
```

```
n <- 200
U <- rnacopula(n, cop)

system.time(theta.hat.beta <- ebeta(U, cop=cop))
theta.hat.beta$root

system.time(theta.hat.dmle <- edmle(U, cop=cop))
theta.hat.dmle$minimum

system.time(theta.hat.etau <- etau(U, cop=cop, method="tau.mean"))
theta.hat.etau

system.time(theta.hat.etau. <- etau(U, cop=cop, method="theta.mean"))
theta.hat.etau.

## etau()  in the case of missing values (NA's)
## ------                 ---------------------
##' @title add Missing Values completely at random
##' @param x  matrix or vector
##' @param prob desired probability ("fraction") of missing values (\code{\link{NA}}s).
##' @return x[] with some (100*prob percent) entries replaced by \code{\link{NA}}s.
addNAs <- function(x, prob) {
    np <- length(x)
    x[sample.int(np, prob*np)] <- NA
    x
}

## UM[] := U[] with 5% missing values
set.seed(7); UM <- addNAs(U, p = 0.05)
mean(is.na(UM)) # 0.05
## This error if x has NA's was happening for  etau(UM, cop=cop)
## before copula version 0.999-17 (June 2017) :
try(eM <- etau(UM, cop=cop, use = "everything"))
        #  --> Error ... NA/NaN/Inf in foreign function call
## The new default:
eM0 <- etau(UM, cop=cop, use = "pairwise")
eM1 <- etau(UM, cop=cop, use = "complete")
##  use = "complete" is really equivalent to dropping all obs. with with missing values:
stopifnot(all.equal(eM1, etau(na.omit(UM), cop=cop), tol = 1e-15))
## but  use = "pairwise" ---> cor(*, use = "pairwise") is much better:
rbind(etau.U = theta.hat.etau, etau.UM.pairwise = eM0, etau.UM.complete = eM1)
```

---

evCopula *Construction of Extreme-Value Copula Objects*

Description

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

Usage

```
evCopula(family, param, dim = 2, ...)
galambosCopula(param)
huslerReissCopula(param)
tawnCopula(param)
tevCopula(param, df = 4, df.fixed = FALSE)
```

Arguments

`family`	a character string specifying the family of an extreme-value copula.
`param`	a numeric vector specifying the parameter values.
`dim`	the dimension of the copula.
`df`	a numerical value specifying the number of degrees of freedom the t extreme-value copula.
`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 `fitCopula`.
`...`	currently nothing.

Value

An object of class "`gumbelCopula`", "`galambosCopula`", "`huslerReissCopula`", "`tawnCopula`", or "`tevCopula`".

Note

The Gumbel copula is both an Archimedean and an extreme-value copula, with principal documentation on `gumbelCopula` (or `archmCopula`).

See Also

`ellipCopula`, `archmCopula`, `gofEVCopula`, `An`.

Examples

```
## Gumbel is both
stopifnot(identical(   evCopula("gumbel"), gumbelCopula()),
          identical(archmCopula("gumbel"), gumbelCopula()))

## For a given degree of dependence these copulas are strikingly similar :

tau <- 1/3

gumbel.cop      <- gumbelCopula     (iTau(gumbelCopula(),      tau))
galambos.cop    <- galambosCopula   (iTau(galambosCopula(),    tau))
huslerReiss.cop <- huslerReissCopula(iTau(huslerReissCopula(), tau))
tawn.cop        <- tawnCopula       (iTau(tawnCopula(),        tau))
tev.cop         <- tevCopula        (iTau(tevCopula(),         tau))
```

```
curve(A(gumbel.cop, x), 0, 1, ylab = "A(<cop>( iTau(<cop>(), tau)), x)",
      main = paste("A(x) for five Extreme Value cop. w/  tau =", format(tau)))
curve(A(galambos.cop, x), lty=2, add=TRUE)
curve(A(huslerReiss.cop, x), lty=3, add=TRUE)
curve(A(tawn.cop, x), lty=4, add=TRUE)
curve(A(tev.cop, x), lty=5, col=2, add=TRUE)# very close to Gumbel

## And look at the differences
curve(A(gumbel.cop, x) - A(tawn.cop, x), ylim = c(-1,1)*0.005,
      ylab = '', main = "A(<Gumbel>, x) - A(<EV-Cop.>, x)")
abline(h=0, lty=2)
curve(A(gumbel.cop, x) - A(galambos.cop, x), add=TRUE, col=2)
curve(A(gumbel.cop, x) - A(huslerReiss.cop, x), add=TRUE, col=3)
curve(A(gumbel.cop, x) - A(tev.cop, x), add=TRUE, col=4, lwd=2)


## the t-EV-copula has always positive tau :
curve(vapply(x, function(x) tau(tevCopula(x)), 0.), -1, 1,
      n=257, ylim=0:1, xlab=quote(rho),ylab=quote(tau),
      main= quote(tau( tevCopula(rho) )), col = 2, lwd = 2)
rect(-1,0,1,1, lty = 2, border = adjustcolor("black", 0.5))
```

---

evCopula-class *Classes Representing Extreme-Value Copulas*

Description

Class evCopula is the virtual (mother) class of all extreme-value copulas. There currently are five subclasses, "galambosCopula", "huslerReissCopula", "tawnCopula", "tevCopula", and "gumbelCopula", the latter of which is also an Archimedean copula, see the page for class "archmCopula".

Objects from the Class

evCopula is a virtual class: No objects may be created from it. Objects of class "galambosCopula" etc, can be created by calls of the form new("galambosCopula", ...), but typically rather by galambosCopula(), etc, see there.

Slots

All slots are inherited from the mother class "copula", see there.

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. Classes "galambosCopula", "huslerReissCopula", "tawnCopula", and "tevCopula" extend class "evCopula" directly.

Note

Objects of class "gumbelCopula" are also of class "archmCopula".

See Also

evCopula, evTestC, evTestK, gofEVCopula, copula-class.

---

evTestA — *Bivariate Test of Extreme-Value Dependence Based on Pickands' Dependence Function*

Description

Test of bivariate extreme-value dependence based on the process comparing the empirical copula with a natural nonparametric estimator of the unknown copula derived under extreme-value dependence. The test statistics are defined in the third reference. Approximate p-values for the test statistics are obtained by means of a *multiplier* technique.

Usage

```
evTestA(x, N = 1000, derivatives = c("An","Cn"),
        ties.method = eval(formals(rank)$ties.method))
```

Arguments

x	a data matrix that will be transformed to pseudo-observations.
N	number of multiplier iterations to be used to simulate realizations of the test statistic under the null hypothesis.
derivatives	string specifying how the derivatives of the unknown copula are estimated, either "An" or "Cn". The former gives better results for samples smaller than 400 but is slower.
ties.method	character string specifying how ranks should be computed if there are ties in any of the coordinate samples of x; passed to pobs.

Details

More details are available in the third reference. See also Genest and Segers (2009) and Remillard and Scaillet (2009).

Value

An object of class htest which is a list, some of the components of which are

statistic	value of the test statistic.
p.value	corresponding approximate p-value.

Note

This test was derived under the assumption of continuous margins, which implies that ties occur with probability zero. The presence of ties in the data might substantially affect the approximate p-value.

References

Genest, C. and Segers, J. (2009). Rank-based inference for bivariate extreme-value copulas. *Annals of Statistics*, 37, pages 2990-3022.

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

Kojadinovic, I. and Yan, J. (2010). Nonparametric rank-based tests of bivariate extreme-value dependence. *Journal of Multivariate Analysis* **101**, 2234–2249.

See Also

evTestK, evTestC, evCopula, gofEVCopula, An.

Examples

```
## Do these data come from an extreme-value copula?
set.seed(63)
uG <- rCopula(100, gumbelCopula(3))
uC <- rCopula(100, claytonCopula(3))
## takes time: 48 seconds on MM's lynne (2012-06)
evTestA(uG)
evTestA(uG, derivatives = "Cn")

evTestA(uC)
```

---

evTestC — *Large-sample Test of Multivariate Extreme-Value Dependence*

Description

Test of multivariate extreme-value dependence based on the empirical copula and max-stability. The test statistics are defined in the second reference. Approximate p-values for the test statistics are obtained by means of a *multiplier* technique.

Usage

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

Arguments

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

Details

More details are available in the second reference. See also Remillard and Scaillet (2009).

Value

An object of `class` `htest` which is a list, some of the components of which are

statistic	value of the test statistic.
p.value	corresponding approximate p-value.

Note

This test was derived under the assumption of continuous margins, which implies that ties occur with probability zero. The presence of ties in the data might substantially affect the approximate p-value.

References

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

Kojadinovic, I., Segers, J., and Yan, J. (2011). Large-sample tests of extreme-value dependence for multivariate copulas. *The Canadian Journal of Statistics* **39**, 4, pages 703-720.

See Also

`evTestK`, `evTestA`, `evCopula`, `gofEVCopula`, `An`.

Examples

```
## Do these data come from an extreme-value copula?
evTestC(rCopula(200, gumbelCopula(3)))
evTestC(rCopula(200, claytonCopula(3)))

## Three-dimensional examples
evTestC(rCopula(200, gumbelCopula(3, dim=3)))
evTestC(rCopula(200, claytonCopula(3, dim=3)))
```

evTestK *Bivariate Test of Extreme-Value Dependence Based on Kendall's Distribution*

Description

Test of extreme-value dependence based on the bivariate probability integral transformation. The test statistic is defined in Ben Ghorbal, G. Nešlehová, and Genest (2009).

Usage

```
evTestK(x, method = c("fsample","asymptotic","jackknife"), ties = NA, N = 100)
```

Arguments

`x`	a data matrix.
`method`	specifies the variance estimation method; can be either `"fsample"` (finite-sample, the default), `"asymptotic"` or `"jackknife"`.
`ties`	logical; if `TRUE`, the original test is adapted to take the presence of ties in the coordinate samples of `x` into account; the default value of `NA` indicates that the presence/absence of ties will be checked for automatically.
`N`	number of samples to be used to estimate a bias term if `ties = TRUE`.

Details

The code for this test was generously provided by Johanna G. Nešlehová. More details are available in Appendix B of Ben Ghorbal, G. Nešlehová and Genest (2009).

Value

An object of `class` `htest` which is a list, some of the components of which are

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

References

Ghorbal, M. B., Genest, C., and G. Nešlehová, J. (2009) On the test of Ghoudi, Khoudraji, and Rivest for extreme-value dependence. *The Canadian Journal of Statistics* **37**, 1–9.

Kojadinovic, I. (2017). Some copula inference procedures adapted to the presence of ties. *Computational Statistics and Data Analysis* **112**, 24–41, `http://arxiv.org/abs/1609.05519`.

See Also

`evTestC`, `evTestA`, `evCopula`, `gofEVCopula`, `An`.

Examples

```
set.seed(321)
## Do the data come from an extreme-value copula?
evTestK(Ug <- rCopula(200, gumbelCopula(3))) # not significant => yes, EV
    dim(Uc <- rCopula(200, claytonCopula(3)))
## Clayton:                        tests are highly significant => no, not EV
(K1 <- evTestK(Uc))
(K2 <- evTestK(Uc, method = "asymptotic"))

system.time(print(K3 <- evTestK(Uc, method = "jackknife")))
## slower: 1.06 sec (2015 intel core i7)
```

---

exchEVTest *Test of Exchangeability for Certain Bivariate Copulas*

Description

Test for assessing the exchangeability of the underlying bivariate copula when it is either extreme-value or left-tail decreasing. The test uses the nonparametric estimators of the Pickands dependence function studied by Genest and Segers (2009).

The test statistic is defined in the second reference. An approximate p-value for the test statistic is obtained by means of a *multiplier* technique if there are no ties in the component series of the bivariate data, or by means of an appropriate bootstrap otherwise.

Usage

```
exchEVTest(x, N = 1000,  estimator = c("CFG", "Pickands"),
            ties = NA, ties.method = eval(formals(rank)$ties.method),
            m = 100, derivatives = c("Cn", "An"))
```

Arguments

`x`	a data matrix that will be transformed to pseudo-observations.
`N`	number of multiplier or boostrap iterations to be used to simulate realizations of the test statistic under the null hypothesis.
`estimator`	string specifying which nonparametric estimator of the Pickands dependence function $A()$ to use; can be either `"CFG"` or `"Pickands"`; see Genest and Segers (2009).
`ties`	logical; if `FALSE`, approximate p-values are computed by means of multiplier bootstrap; if `TRUE`, a boostrap adapted to the presence of ties in any of the coordinate samples of `x` is used; the default value of `NA` indicates that the presence/absence of ties will be checked for automatically.
`ties.method`	string specifying how ranks should be computed if there are ties in any of the coordinate samples of `x`; passed to `pobs`.

`derivatives` a string specifying how the derivatives of the unknown copula are estimated; can be either "An" or "Cn". The former should be used under the assumption of extreme-value dependence. The latter is faster; see the second reference.

`m` integer specifying the size of the integration grid for the statistic.

Details

More details are available in the references.

Value

An object of class `htest` which is a list, some of the components of which are

`statistic` value of the test statistic.

`pvalue` corresponding approximate p-value.

References

Genest, C. and Segers, J. (2009) Rank-based inference for bivariate extreme-value copulas. *Annals of Statistics* **37**, 2990–3022.

Kojadinovic, I. and Yan, J. (2012) A nonparametric test of exchangeability for extreme-value and left-tail decreasing bivariate copulas. *The Scandinavian Journal of Statistics* **39:3**, 480–496.

Kojadinovic, I. (2017). Some copula inference procedures adapted to the presence of ties. *Computational Statistics and Data Analysis* **112**, 24–41, http://arxiv.org/abs/1609.05519.

See Also

`exchTest`, `radSymTest`, `gofCopula`.

Examples

```
## Data from an exchangeable left-tail decreasing copulas
exchEVTest(rCopula(200,  gumbelCopula(3)))
exchEVTest(rCopula(200, claytonCopula(3)))

## An asymmetric Khoudraji-Gumbel-Hougaard copula
kc <- khoudrajiCopula(copula1 = indepCopula(),
                      copula2 = gumbelCopula(4),
                      shapes = c(0.4, 0.95))
exchEVTest(rCopula(200, kc))
```

exchTest *Test of Exchangeability for a Bivariate Copula*

Description

Test for assessing the exchangeability of the underlying bivariate copula based on the empirical copula. The test statistics are defined in the first two references. Approximate p-values for the test statistics are obtained by means of a *multiplier* technique if there are no ties in the component series of the bivariate data, or by means of an appropriate bootstrap otherwise.

Usage

```
exchTest(x, N = 1000, ties = NA, m = 0)
```

Arguments

`x`	a data matrix that will be transformed to pseudo-observations.
`N`	number of multiplier or boostrap iterations to be used to simulate realizations of the test statistic under the null hypothesis.
`ties`	logical; if `FALSE`, approximate p-values are computed by means of multiplier bootstrap; if `TRUE`, a boostrap adapted to the presence of ties in any of the coordinate samples of `x` is used; the default value of `NA` indicates that the presence/absence of ties will be checked for automatically.
`m`	if `m=0`, integration in the Cramér–von Mises statistic is carried out with respect to the empirical copula; if `m > 0`, integration is carried out with respect to the Lebesgue measure and `m` specifies the size of the integration grid.

Details

More details are available in the references.

Value

An object of `class` `htest` which is a list, some of the components of which are

`statistic`	value of the test statistic.
`p.value`	corresponding approximate p-value.

References

Genest, C., G. Nešlehová, J. and Quessy, J.-F. (2012). Tests of symmetry for bivariate copulas. *Annals of the Institute of Statistical Mathematics* **64**, 811–834.

Kojadinovic, I. and Yan, J. (2012). A nonparametric test of exchangeability for extreme-value and left-tail decreasing bivariate copulas. *The Scandinavian Journal of Statistics* **39:3**, 480–496.

Kojadinovic, I. (2017). Some copula inference procedures adapted to the presence of ties. *Computational Statistics and Data Analysis* **112**, 24–41, `http://arxiv.org/abs/1609.05519`.

See Also

radSymTest, exchEVTest, gofCopula.

Examples

```
## Data from an exchangeable copulas
exchTest(rCopula(200,  gumbelCopula(3)))
exchTest(rCopula(200, claytonCopula(3)))

## An asymmetric Khoudraji-Clayton copula
kc <- khoudrajiCopula(copula1 = indepCopula(),
                      copula2 = claytonCopula(6),
                      shapes = c(0.4, 0.95))
exchTest(rCopula(200, kc))
```

---

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

param	a numeric vector specifying the parameter values.
dim	the dimension of the copula.
...	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

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

See Also

Copula, copula-class, fitCopula.

Examples

```
## a bivariate example
fgm.cop <- fgmCopula(1)
x <- rCopula(1000, fgm.cop)
cor(x, method = "kendall")
tau(fgm.cop)
cor(x, method = "spearman")
rho(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()`.

`subsets.char`: Object of class `"character"`, containing the subsets of integers used for naming the parameters.

`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.

`fullname`: 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

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

See Also

`copula-class`, `fgmCopula-class`.

`fitCopula` *Fitting Copulas to Data – Copula Parameter Estimation*

Description

Parameter estimation of copulas, i.e., fitting of a copula model to multivariate (possibly “pseudo”) observations.

Usage

```
loglikCopula(param, u, copula)

## Generic [and "rotCopula" method] :
fitCopula(copula, data, ...)
## S4 method for signature 'copula'
fitCopula(copula, data,
          method = c("mpl", "ml", "itau", "irho", "itau.mpl"),
          posDef = is(copula, "ellipCopula"),
          start = NULL, lower = NULL, upper = NULL,
          optim.method = optimMeth(copula, method, dim = d),
          optim.control = list(maxit=1000),
          estimate.variance = NA, hideWarnings = FALSE, ...)

optimMeth(copula, method, dim)
```

Arguments

`param` vector of parameter values.

`u` $n \times d$-matrix of (pseudo-)observations in $[0, 1]^d$ for computing the copula log-likelihood, where n denotes the sample size and d the dimension. Consider applying the function `pobs()` first in order to obtain such data.

`data` as u, an $n \times d$-matrix of data. For `method` being `"mpl"`, `"ml"` or `"itau.mpl"`, this has to be data in $[0, 1]^d$. For `method` being `"itau"` or `"irho"`, it can either be data in $[0, 1]^d$ or in the whole d-dimensional space.

`copula` a `"copula"` object.

`method` a `character` string specifying the copula parameter estimator used. This can be one of:

- **"mpl"** Maximum pseudo-likelihood estimator (based on "pseudo-observations" in $[0, 1]^d$, typical obtained via `pobs()`).
- **"ml"** As `"mpl"` just with a different variance estimator. For this to be correct (thus giving the true MLE), `data` are assumed to be observations from the true underlying copula whose parameter is to be estimated.
- **"itau"** Inversion of Kendall's tau estimator. `data` can be either in $[0, 1]^d$ (true or pseudo-observations of the underlying copula to be estimated) or in the d-dimensional space.
- **"irho"** As `"itau"` just with Spearman's rho instead of Kendall's tau.
- **"itau.mpl"** This is the estimator of t copula parameters suggested by Mashal and Zeevi (2002) (see also Demarta and McNeil (2005)) based on the given `data` in $[0, 1]^d$ (true or pseudo-observations of the underlying copula to be estimated). Note that it requires `dispstr = "un"`.

`posDef` a `logical` indicating whether a proper correlation matrix is computed.

`start` a `vector` of starting values for the parameter optimization via `optim()`.

`lower, upper` Lower or upper parameter bounds for the optimization methods `"Brent"` or `"L-BFGS-B"`.

`optim.control` a `list` of control parameters passed to `optim(*, control=optim.control)`.

`optim.method` a character string specify the optimization method *or* a `function` which when called with arguments `(copula, method, dim)` will return such a character string, see `optim()`'s `method`; only used when `method = "mpl"` or `"ml"`.

The default has been changed (for **copula** 0.999-16, in Aug. 2016) from `"BFGS"` to the result of `optimMeth(copula, method, dim)` which is often `"L-BFGS-B"`.

`dim` integer, the data and copula dimension, $d \geq 2$.

`estimate.variance`
a `logical` indicating whether the estimator's asymptotic variance is computed (if available for the given `copula`; the default `NA` computes it for the `methods` `"itau"` and `"irho"`, cannot (yet) compute it for `"itau.mpl"` and only computes it for `"mpl"` or `"ml"` if the optimization converged).

`hideWarnings` a `logical`, which, if `TRUE`, suppresses warnings from the involved likelihood maximization (typically when the likelihood is evaluated at invalid parameter values).

... additional arguments passed to `method` specific auxiliary functions, e.g., `traceOpt = TRUE` for tracing `optimize` for method `"itau.mpl"`, and for “"manual"” tracing with method `"ml"` or `"mpl"` (notably for `optim.method="Brent"`).

Details

The only difference between `"mpl"` and `"ml"` is in the variance-covariance estimate, *not* in the parameter (θ) estimates.

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. Standard errors are computed as explained in Genest, Ghoudi and Rivest (1995); see also Kojadinovic and Yan (2010, Section 3). Their estimation requires the computation of certain partial derivatives of the (log) density. These have been implemented for six copula families thus far: the Clayton, Gumbel-Hougaard, Frank, Plackett, normal and t copula families. For other families, numerical differentiation based on `grad()` from package **numDeriv** is used (and a warning message is displayed).

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. In that case, `nearPD(*, corr=TRUE)` (from **Matrix**) is applied to get a proper correlation matrix.

For normal and t copulas, `fitCopula(, method = "mpl")` and `fitCopula(, method = "ml")` maximize the log-likelihood based on **mvtnorm**'s `dmvnorm()` and `dmvt()`, respectively. The latter two functions set the respective densities to zero if the correlation matrices of the corresponding distributions are not positive definite. As such, the estimated correlation matrices will be positive definite.

If methods `"itau"` or `"irho"` are used in `fitCopula()`, an estimate of the asymptotic variance (if available for the copula under consideration) will be correctly computed only if the argument `data` consists of pseudo-observations (see `pobs()`).

Consider the t copula with `df.fixed=FALSE` (see `ellipCopula()`). In this case, the methods `"itau"` and `"irho"` cannot be used in `fitCopula()` as they cannot estimate the degrees of freedom parameter `df`. For the methods `"mpl"` and `"itau.mpl"` the asymptotic variance cannot be (fully) estimated (yet). For the methods `"ml"` and `"mpl"`, when `start` is not specified, the starting value for `df` is set to `copula@df`, typically 4.

To implement the *Inference Functions for Margins* (IFM) method (see, e.g., Joe 2005), set `method="ml"` and note that `data` need to be parametric pseudo-observations obtained from *fitted* parametric marginal distribution functions. The returned large-sample variance will then underestimate the true variance (as the procedure cannot take into account the (unknown) estimation error for the margins).

The fitting procedures based on `optim()` generate warnings because invalid parameter values are tried during the optimization process. When the number of parameters is one and the parameter space is bounded, using `optim.method="Brent"` is likely to give less warnings. Furthermore, from experience, `optim.method="Nelder-Mead"` is sometimes a more robust alternative to `optim.method="BFGS"` or `"L-BFGS-B"`.

There are methods for `vcov()`, `coef()`, `logLik()`, and `nobs()`.

Value

`loglikCopula()` returns the copula log-likelihood evaluated at the parameter (vector) `param` given the data `u`.

The return value of `fitCopula()` is an object of class `"fitCopula"` (inheriting from hidden class `"fittedMV"`), containing (among others!) the slots

estimate The parameter estimates.

var.est The large-sample (i.e., asymptotic) variance estimate of the parameter estimator; (filled with) `NA` if `estimate.variance=FALSE`.

copula The fitted copula object.

The `summary()` method for `"fitCopula"` objects returns an S3 "class" `"summary.fitCopula"`, which is simply a list with components `method`, `loglik` and `convergence`, all three from the corresponding slots of the `"fitCopula"` objects, and `coefficients` (a matrix of estimated coefficients, standard errors, t values and p-values).

References

Genest, C. (1987). Frank's family of bivariate distributions. *Biometrika* **74**, 549–555.

Genest, C. and Rivest, L.-P. (1993). Statistical inference procedures for bivariate Archimedean copulas. *Journal of the American Statistical Association* **88**, 1034–1043.

Rousseeuw, P. and Molenberghs, G. (1993). Transformation of nonpositive semidefinite correlation matrices. *Communications in Statistics: Theory and Methods* **22**, 965–984.

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

Joe, H. (2005). Asymptotic efficiency of the two-stage estimation method for copula-based models. *Journal of Multivariate Analysis* **94**, 401–419.

Mashal, R. and Zeevi, A. (2002). Beyond Correlation: Extreme Co-movements Between Financial Assets. https://www0.gsb.columbia.edu/faculty/azeevi/PAPERS/BeyondCorrelation.pdf (2016-04-05)

Demarta, S. and McNeil, A. J. (2005). The t copula and related copulas. *International Statistical Review* **73**, 111–129.

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

Kojadinovic, I. and Yan, J. (2010). Comparison of three semiparametric methods for estimating dependence parameters in copula models. *Insurance: Mathematics and Economics* **47**, 52–63.

See Also

`Copula`, `fitMvdc` for fitting multivariate distributions *including* the margins, `gofCopula` for goodness-of-fit tests.

For maximum likelihood of (nested) Archimedean copulas, see `emle`, etc.

Examples

```
(Xtras <- copula:::doExtras()) # determine whether examples will be extra (long)
n <- if(Xtras) 200 else 64 # sample size

## A Gumbel copula
set.seed(7) # for reproducibility
```

```
gumbel.cop <- gumbelCopula(3, dim=2)
x <- rCopula(n, gumbel.cop) # "true" observations (simulated)
u <- pobs(x)                # pseudo-observations
## Inverting Kendall's tau
fit.tau <- fitCopula(gumbelCopula(), u, method="itau")
fit.tau
confint(fit.tau) # work fine !
confint(fit.tau, level = 0.98)
summary(fit.tau) # a bit more, notably "Std. Error"s
coef(fit.tau)# named vector
coef(fit.tau, SE = TRUE)# matrix

## Inverting Spearman's rho
fit.rho <- fitCopula(gumbelCopula(), u, method="irho")
summary(fit.rho)
## Maximum pseudo-likelihood
fit.mpl <- fitCopula(gumbelCopula(), u, method="mpl")
fit.mpl
## Maximum likelihood -- use 'x', not 'u' ! --
fit.ml <- fitCopula(gumbelCopula(), x, method="ml")
summary(fit.ml) # now prints a bit more than simple 'fit.ml'
## ... and what's the log likelihood (in two different ways):
(ll. <- logLik(fit.ml))
stopifnot(all.equal(as.numeric(ll.),
            loglikCopula(coef(fit.ml), u=x, copula=gumbel.cop)))

## A Gauss/normal copula

## With multiple/*un*constrained parameters
set.seed(6) # for reproducibility
normal.cop <- normalCopula(c(0.6, 0.36, 0.6), dim=3, dispstr="un")
x <- rCopula(n, normal.cop) # "true" observations (simulated)
u <- pobs(x)                # pseudo-observations
## Inverting Kendall's tau
fit.tau <- fitCopula(normalCopula(dim=3, dispstr="un"), u, method="itau")
fit.tau
## Inverting Spearman's rho
fit.rho <- fitCopula(normalCopula(dim=3, dispstr="un"), u, method="irho")
fit.rho
## Maximum pseudo-likelihood
fit.mpl <- fitCopula(normalCopula(dim=3, dispstr="un"), u, method="mpl")
summary(fit.mpl)
coef(fit.mpl) # named vector
coef(fit.mpl, SE = TRUE) # the matrix, with SE
## Maximum likelihood (use 'x', not 'u' !)
fit.ml <- fitCopula(normalCopula(dim=3, dispstr="un"), x, method="ml")
summary(fit.ml)
confint(fit.ml)
confint(fit.ml, level = 0.999) # clearly non-0

## Fix some of the parameters
param <- c(.6, .3, NA_real_)
attr(param, "fixed") <- c(TRUE, FALSE, FALSE)
```

```
ncp <- normalCopula(param = param, dim = 3, dispstr = "un")
fixedParam(ncp) <- c(TRUE, TRUE, FALSE)
summary(Fxf.mpl <- fitCopula(ncp, u, method = "mpl"))
Fxf.mpl@copula # reminding of the fixed param. values

## With dispstr = "toep" :
normal.cop.toep <- normalCopula(c(0, 0), dim=3, dispstr="toep")
## Inverting Kendall's tau
fit.tau <- fitCopula(normalCopula(dim=3, dispstr="toep"), u, method="itau")
fit.tau
## Inverting Spearman's rho
fit.rho <- fitCopula(normalCopula(dim=3, dispstr="toep"), u, method="irho")
summary(fit.rho)
## Maximum pseudo-likelihood
fit.mpl <- fitCopula(normalCopula(dim=3, dispstr="toep"), u, method="mpl")
fit.mpl
## Maximum likelihood (use 'x', not 'u' !)
fit.ml <- fitCopula(normalCopula(dim=3, dispstr="toep"), x, method="ml")
summary(fit.ml)

## With dispstr = "ar1"
normal.cop.ar1 <- normalCopula(c(0), dim=3, dispstr="ar1")
## Inverting Kendall's tau
summary(fit.tau <- fitCopula(normalCopula(dim=3, dispstr="ar1"), u, method="itau"))
## Inverting Spearman's rho
summary(fit.rho <- fitCopula(normalCopula(dim=3, dispstr="ar1"), u, method="irho"))
## Maximum pseudo-likelihood
summary(fit.mpl <- fitCopula(normalCopula(dim=3, dispstr="ar1"), u, method="mpl"))
## Maximum likelihood (use 'x', not 'u' !)
fit.ml <- fitCopula(normalCopula(dim=3, dispstr="ar1"), x, method="ml")
summary(fit.ml)

## A t copula with variable df (df.fixed=FALSE)
(tCop <- tCopula(c(0.2,0.4,0.6), dim=3, dispstr="un", df=5))
set.seed(101)
x <- rCopula(n, tCop) # "true" observations (simulated)
## Maximum likelihood (start = (rho[1:3], df))
summary(tc.ml <- fitCopula(tCopula(dim=3, dispstr="un"), x, method="ml",
                           start = c(0,0,0, 10)))
## Maximum pseudo-likelihood (the asymptotic variance cannot be estimated)
u <- pobs(x)          # pseudo-observations
tc.mpl <- fitCopula(tCopula(dim=3, dispstr="un"),
                     u, method="mpl", estimate.variance=FALSE,
                     start= c(0,0,0, 10))
summary(tc.mpl)
```

---

fitCopula-class *Classes of Fitted Multivariate Models: Copula, Mvdc*

Description

Classes and summary methods related to copula model fitting.

Objects from the Class

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

Slots

The “mother class”, `"fittedMV"` has the slots

`estimate`: `numeric`, the estimated parameters.

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

`loglik`: `numeric`, log likelihood evaluated at the maximizer.

`nsample`: `numeric`, integer representing the sample size.

`method`: `character`, method of estimation.

`fitting.stats`: a `list`, currently containing the numeric `convergence` code from `optim`, the `counts`, `message`, and all the `control` arguments explicitly passed to `optim`.

In addition, the `"fitCopula"` class has a slot

`copula`: the *fitted* copula, of class `"copula"`.

whereas the `"fitMvdc"` has

`mvdc`: the *fitted* distribution, of class `"mvdc"`.

Extends

Classes `"fitCopula"` and `"fitMvdc"` extend class `"fittedMV"`, directly.

Methods

summary `signature(object = "fitMvdc")`: ...

summary `signature(object = "fitCopula")`: ...

Further, there are S3 methods (class `"fittedMV"`) for `coef()`, `vcov()` and `logLik()`, see `fitMvdc`.

References

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

fitLambda *Non-parametric Estimators of the Matrix of Tail-Dependence Coefficients*

Description

Computing non-parametric estimators of the (matrix of) tail-dependence coefficients.

Usage

```
fitLambda(u, method = c("Schmid.Schmidt", "t"), p = min(100/nrow(u), 0.1),
          lower.tail = TRUE, verbose = FALSE, ...)
```

Arguments

`u`	$n \times d$-`matrix` of (pseudo-)observations in $[0, 1]^d$ for estimating the (matrix of) tail-dependence coefficients.
`method`	the method with which the tail-dependence coefficients are computed: `method = "Schmid.Schmidt"`: nonparametric estimator of Schmid and Schmidt (2007) (see also Jaworksi et al. (2009, p. 231)) computed for all pairs. `method = "t"`: fits pairwise t copulas and returns the implied tail-dependence coefficient.
`p`	(small) cut-off parameter in $[0, 1]$ below (for `tail = "lower"`) or above (for `tail = "upper"`) which the estimation takes place.
`lower.tail`	`logical` indicating whether the lower (the default) or upper tail-dependence coefficient is computed.
`verbose`	a `logical` indicating whether a progress bar is displayed.
`...`	additional arguments passed to the underlying functions (at the moment only to `optimize()` in case `method = "t"`).

Details

As seen in the examples, be careful using nonparametric estimators, they might not perform too well (depending on `p` and in general). After all, the notion of tail dependence is a limit, finite sample sizes might not be able to capture well.

Value

The matrix of pairwise coefficients of tail dependence; for `method = "t"` a list additional containing the matrix of pairwise estimated correlation coefficients and the matrix of pairwise estimated degrees of freedom.

References

Jaworski, P., Durante, F., Härdle, W. K., Rychlik, T. (2010). *Copula Theory and Its Applications* Springer, Lecture Notes in Statistics – Proceedings.

Schmid, F., Schmidt, R. (2007). Multivariate conditional versions of Spearman's rho and related measures of tail dependence. *Journal of Multivariate Analysis* **98**, 1123–1140.

Examples

```
n <- 10000 # sample size
p <- 0.01 # cut-off

## Bivariate case
d <- 2
cop <- claytonCopula(2, dim = d)
set.seed(271)
U <- rCopula(n, copula = cop) # generate observations (unrealistic)
(lam.true <- lambda(cop)) # true tail-dependence coefficients lambda
(lam.C <- c(lower = fitLambda(U, p = p)[2,1],
            upper = fitLambda(U, p = p, lower.tail = FALSE)[2,1])) # estimate lambdas
## => pretty good
U. <- pobs(U) # pseudo-observations (realistic)
(lam.C. <- c(lower = fitLambda(U., p = p)[2,1],
             upper = fitLambda(U., p = p, lower.tail = FALSE)[2,1])) # estimate lambdas
## => The pseudo-observations do have an effect...

## Higher-dimensional case
d <- 5
cop <- claytonCopula(2, dim = d)
set.seed(271)
U <- rCopula(n, copula = cop) # generate observations (unrealistic)
(lam.true <- lambda(cop)) # true tail-dependence coefficients lambda
(Lam.C <- list(lower = fitLambda(U, p = p),
               upper = fitLambda(U, p = p, lower.tail = FALSE))) # estimate Lambdas
## => Not too good
U. <- pobs(U) # pseudo-observations (realistic)
(Lam.C. <- list(lower = fitLambda(U., p = p),
                upper = fitLambda(U., p = p, lower.tail = FALSE))) # estimate Lambdas
## => Performance not too great here in either case
```

`fitMvdc` *Estimation of Multivariate Models Defined via Copulas*

Description

Fitting copula-based multivariate distributions ("`mvdc`") to multivariate data, estimating both the marginal and the copula parameters.

If you assume non parametric margins, in other words, take the empirical distributions for all margins, you can use `fitCopula(*, pobs(x))` instead.

Usage

```
loglikMvdc(param, x, mvdc)
fitMvdc(data, mvdc, start, optim.control = list(), method = "BFGS",
        lower = -Inf, upper = Inf,
        estimate.variance = fit$convergence == 0, hideWarnings = TRUE)

## S3 method for class 'fittedMV'
coef(object, SE = FALSE, ...)
## S3 method for class 'fittedMV'
logLik(object, ...)
## S3 method for class 'fittedMV'
vcov(object, ...)
```

Arguments

`param`	a vector of parameter values. When specifying parameters for `mvdc` objects, the parameters must be ordered with the marginals first and the copula parameters last. When the `mvdc` object has `marginsIdentical = TRUE`, only the parameters of one marginal must be specified.
`x`	a data matrix.
`mvdc`	a "`mvdc`" object.
`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`.
`lower, upper`	bounds on each parameter, passed to `optim`, typically "box constraints" for `method = "L-BFGS-B"`.
`estimate.variance`	logical; if true (as by default, if the optimization converges), the asymptotic variance is estimated.
`hideWarnings`	logical indicating if warning messages from likelihood maximization, e.g., from evaluating at invalid parameter values, should be suppressed (via `suppressWarnings`).
`object`	an R object of class "`fitMvdc`".
`SE`	for the `coef` method, a logical indicating if standard errors should be returned in addition to the estimated parameters (in a `matrix`). This is equivalent, but more efficient than, e.g., `coef(summary(object))`.
`...`	potentially further arguments to methods.

Value

The return value `loglikMvdc()` is the log likelihood evaluated for the given value of `param`.

The return value of `fitMvdc()` is an object of class "`fitMvdc`" (see there), containing slots (among others!):

`estimate` the estimate of the parameters.

`var.est` large-sample (i.e., asymptotic) variance estimate of the parameter estimator (filled with `NA` if `estimate.variance = FALSE`).

`mvdc` the *fitted* multivariate distribution, see `mvdc`.

The `summary()` method for `"fitMvdc"` objects returns a S3 "class" `"summary.fitMvdc"`, simply a list with components `method`, `loglik`, and `convergence`, all three from corresponding slots of the `"fitMvdc"` objects, and

`coefficients` a matrix of estimated coefficients, standard errors, t values and p-values.

Note

User-defined marginal distributions can be used as long as the `"{dpq}"` functions are defined. See `vignette("AR_Clayton", package="copula")`.

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 some of the fitting functions generate error messages because invalid parameter values are tried during the optimization process (see `optim`). This should be rarer since 2013, notably for likelihood based methods (as the likelihood is now rather set to `-Inf` than giving an error).

Previously, `loglikMvdc()` had an argument `hideWarnings`; nowadays, do use `suppressWarnings(..)` if you are sure you do not want to see them.

See Also

`mvdc` and `mvdc`; further, `Copula`, `fitCopula`, `gofCopula`.

For fitting univariate marginals, `fitdistr()`.

Examples

```
G3 <- gumbelCopula(3, dim=2)
gMvd2  <- mvdc(G3, c("exp","exp"),
               param = list(list(rate=2), list(rate=4)))
## with identical margins:
gMvd.I <- mvdc(G3, "exp",
               param = list(rate=3), marginsIdentical=TRUE)

(Xtras <- copula:::doExtras()) # determine whether examples will be extra (long)
n <- if(Xtras) 10000 else 200 # sample size (realistic vs short for example)

set.seed(11)
x <- rMvdc(n, gMvd2)
## Default:     hideWarnings = FALSE .. i.e. show warnings here
fit2 <- fitMvdc(x, gMvd2, start = c(1,1, 2))
fit2
confint(fit2)
summary(fit2) # slightly more
## The estimated, asymptotic var-cov matrix [was used for confint()]:
vcov(fit2)
```

```
## with even more output for the "identical margin" case:
fitI <- fitMvdc(x, gMvd.I, start = c(3, 2),
                optim.control=list(trace= TRUE, REPORT= 2))
summary(fitI)
coef(fitI, SE = TRUE)
stopifnot(is.matrix(coef(fitI, SE = TRUE)),
          is.matrix(print( confint(fitI) )) )

## a wrong starting value can already be *the* problem:
f2 <- try(fitMvdc(x, gMvd.I, start = c(1, 1),
           optim.control=list(trace= TRUE, REPORT= 2)))
##--> Error in optim( ... ) : non-finite finite-difference value [2]

##==> "Solution" :  Using a more robust (but slower) optim() method:
fI.2 <- fitMvdc(x, gMvd.I, start = c(1, 1), method = "Nelder",
                optim.control=list(trace= TRUE))
fI.2
```

`fixParam` *Fix a Subset of a Copula Parameter Vector*

Description

It is sometimes useful to keep fixed some components of a copula parameter vector whereas the others are "free" and will be estimated, e.g., by `fitCopula`.

The first two functions set or modify the "fixedness", whereas `isFree()`, `isFreeP()` and `nParam()` are utilities enquiring about the "fixedness" of the parameters (of a copula).

Usage

```
fixParam(param, fixed = TRUE)
fixedParam(copula) <- value

isFreeP(param)
## S4 method for signature 'copula'
isFree(copula)
## and specific '*Copula' methods
## S4 method for signature 'copula'
nParam(copula, freeOnly = FALSE)
## and specific '*Copula' methods
```

Arguments

`param`	numeric parameter vector
`fixed, value`	logical vector of the same length as `param` indicating for each component `param[j]` if it is (going to be) fixed or not.

`copula`	a "`copula`" object.
`freeOnly`	logical (scalar) indicating if only free parameters should be counted or all.

Value

`fixParam(param)` returns a `numeric` vector with attribute `"fixed"`(a `logical`, either `TRUE` or vector of the same length as `param`) to indicate which components of `param` are to be held fixed or not.

`fixedParam<-`, a generic function, returns a "`copula`" object with a partly fixed parameter (slot), i.e., corresponding to `fixParam()` above.

See Also

`fitCopula` for fitting; t-copulas, `tCopula(*, df.fixed=TRUE)` now uses parameter fixing for `"df"`.

`setTheta()` for setting or *changing* the *non-fixed* parameter values.

Examples

```
nc1 <- normalCopula(dim = 3, fixParam(c(.6,.3,.2), c(TRUE, FALSE,FALSE)),
                    dispstr = "un")
nc1
nc13 <- nc12 <- nc1
fixedParam(nc12) <- c(TRUE, TRUE, FALSE) ; nc12
fixedParam(nc13) <- c(TRUE, FALSE, TRUE) ; nc13
set.seed(17); x <- rCopula(100, nc1)
summary(f.13 <- fitCopula(nc13, x, method = "ml"))
f.13@copula  # 'rho.2' is estimated; the others kept fixed

## Setting to 'FALSE' (removes the "fixed" parts completely):
nc0 <- nc13; fixedParam(nc0) <- FALSE
nc0
stopifnot(is.null(attributes(nc0@parameters)))
```

`gasoil` *Daily Crude Oil and Natural Gas Prices from 2003 to 2006*

Description

Three years of daily prices (from July 2003 to July 2006) of crude oil and natural gas. These data should be very close to those analysed in Grégoire, Genest and Gendron (2008).

Usage

```
data(gasoil)
```

Format

A data frame of 762 daily prices from 2003 to 2006.

`date` date (of class `Date`).

`oil` daily price of crude oil

`gas` daily price of natural gas

References

Grégoire, V., Genest, C., and Gendron, M. (2008) Using copulas to model price dependence in energy markets. *Energy Risk* **5**(5), 58–64.

Examples

```
data(gasoil)
## Log Scaled  Oil & Gas Prices :
lattice :: xyplot(oil + gas ~ date, data = gasoil, auto.key=TRUE,
                  type = c("l","r"),
                  scales=list(y = list(log = TRUE), equispaced.log = FALSE))
```

`generator` *Generator Functions for Archimedean and Extreme-Value Copulas*

Description

Methods to evaluate the generator function, the inverse generator function, and derivatives of the inverse of the generator function for Archimedean copulas. For extreme-value copulas, the "Pickands dependence function" plays the role of a generator function.

Usage

```
psi(copula, s)

iPsi(copula, u, ...)
diPsi(copula, u, degree=1, log=FALSE, ...)

A(copula, w)
dAdu(copula, w)
```

Arguments

`copula`	an object of class "`copula`".
`u, s, w`	numerical vector at which these functions are to be evaluated.
`...`	further arguments for specific families.
`degree`	the degree of the derivative (defaults to 1).
`log`	logical indicating if the `log` of the *absolute* derivative is desired. Note that the derivatives of psi alternate in sign.

Details

`psi()` and `iPsi()` are, respectively, the generator function $\psi()$ and its inverse $\psi^{(-1)}$ for an Archimedean copula, see `pnacopula` for definition and more details.

`diPsi()` computes (currently only the first two) derivatives of `iPsi()` ($= \psi^{(-1)}$).

`A()`, the "Pickands dependence function", can be seen as the generator function of an extreme-value copula. For instance, in the bivariate case, we have the following result (see, e.g., Gudendorf and Segers 2009):

A bivariate copula C is an extreme-value copula if and only if

$$C(u,v) = (uv)^{A(\log(v)/\log(uv))}, \qquad (u,v) \in (0,1]^2 \setminus \{(1,1)\},$$

where $A : [0,1] \to [1/2,1]$ is convex and satisfies $\max(t, 1-t) \le A(t) \le 1$ for all $t \in [0,1]$.
In the d-variate case, there is a similar characterization, except that this time, the Pickands dependence function A is defined on the d-dimensional unit simplex.

`dAdu()` returns a data.frame containing the 1st and 2nd derivative of `A()`.

References

Gudendorf, G. and Segers, J. (2010). Extreme-value copulas. In *Copula theory and its applications*, Jaworski, P., Durante, F., Härdle, W. and Rychlik, W., Eds. Springer-Verlag, Lecture Notes in Statistics, 127–146, http://arxiv.org/abs/0911.1015.

See Also

Nonparametric estimators for $A()$ are available, see `An`.

Examples

```
## List the available methods (and their definitions):
showMethods("A")
showMethods("iPsi", incl=TRUE)
```

---

getAcop — *Get "acopula" Family Object by Name*

Description

Get one of our "acopula" family objects (see acopula-families by name.

Named strings for "translation" between different names and forms of Archimedean copulas.

Usage

```
getAcop (family, check = TRUE)
getAname(family, objName = FALSE)

.ac.shortNames
.ac.longNames
.ac.objNames
.ac.classNames
```

Arguments

family
: either a character string, the short or longer form of the Archimedean family name (for example, "Clayton" or simply "C"; see the acopula-families documentation), or an acopula family object, or an object inheriting from class archmCopula.

check
: logical indicating whether the class of the return value should be checked to be "acopula".

objName
: logical indicating that the *name* of the R object should be returned, instead of the family name, e.g., "copClayton" instead of "Clayton".

Value

getAcop() returns an "acopula" family object, typically one of one of our predefined ones.

getAname() returns a character string, the name of an "acopula" family object.

.as.longnames etc are named string constants, useful in programming for all our (five) standard Archimedean families.

See Also

Our predefined acopula-families; the class definition "acopula".

Examples

```
getAcop("Gumbel")

## different ways of getting the same "acopula" family object:
stopifnot(## Joe (three ways):
          identical(getAcop("J"), getAcop("Joe")),
          identical(getAcop("J"), copJoe),
          ## Frank (yet another two different ways):
          identical(getAcop(frankCopula()), copFrank),
          identical(getAcop("frankCopula"), copFrank))

stopifnot(
  identical(getAname(claytonCopula()), getAname("C")),
  identical(getAname(copClayton), "Clayton"), identical(getAname("J"), "Joe"),
  identical(getAname(amhCopula(), TRUE), "copAMH"),
  identical(getAname(joeCopula(), TRUE), "copJoe")
)

.ac.shortNames
.ac.longNames
.ac.objNames
.ac.classNames
```

getTheta *Get the Parameter(s) of a Copula*

Description

Get the parameter (vector) θ (theta) of a copula, see setTheta for more background.

Usage

```
getTheta(copula, freeOnly = TRUE, attr = FALSE, named = attr)
```

Arguments

copula	an R object of class copula.
freeOnly	logical indicating that only non-fixed aka "free" parameters are to be returned (as vector).
attr	logical indicating if attributes (such as lower and uppder bounds for each parameters) are to be returned as well.
named	logical if the resulting parameter vector should have names.

Value

parameter vector of the copula, a numeric vector, possibly with names and other attributes (depending on the attr and named arguments).

Methods

signature(copula = "copula") ..

signature(copula = "acopula") ..

signature(copula = "khoudrajiCopula") ..

signature(copula = "mixCopula") ..

signature(copula = "rotCopula") ..

signature(copula = "xcopula") ..

See Also

setTheta, its inverse.

Examples

```
getTheta(setTheta(copClayton, 0.5)) # is 0.5
```

ggraph-tools	*Computations for Graphical GOF Test via Pairwise Rosenblatt Transforms*

Description

Tools for computing a graphical goodness-of-fit (GOF) test based on pairwise Rosenblatt transformed data.

`pairwiseCcop()` computes a (n, d, d)-`array` which contains pairwise Rosenblatt-transformed data.

`pairwiseIndepTest()` takes such an array as input and computes a (d, d)-`matrix` of test results from pairwise tests of independence (as by `indepTest()`).

`pviTest()` can be used to extract the matrix of p-values from the return matrix of `pairwiseIndepTest()`.

`gpviTest()` takes such a matrix of p-values and computes a global p-value with the method provided.

Usage

```
pairwiseCcop(u, copula, ...)
pairwiseIndepTest(cu.u, N=256,
        iTest = indepTestSim(n, p=2, m=2, N=N, verbose = idT.verbose, ...),
        verbose=TRUE, idT.verbose = verbose, ...)

 pviTest(piTest)
gpviTest(pvalues, method=p.adjust.methods, globalFun=min)
```

Arguments

`u`	(n, d)-`matrix` of copula data.
`copula`	copula object used for the Rosenblatt transform (H_0 copula).
`...`	additional arguments passed to the internal function which computes the conditional copulas (for `pairwiseCcop()`). Can be used to pass, for example, the degrees of freedom parameter `df` for t-copulas. For `pairwiseIndepTest()`, ... are passed to `indepTestSim()`.
`cu.u`	(n, d, d)-`array` as returned by `pairwiseCcop()`.
`N`	argument of `indepTestSim()`.
`iTest`	the result of (a version of) `indepTestSim()`; as it does *not* depend on the data, and is costly to compute, it can be computed separately and passed here.
`verbose`	`integer` (or `logical`) indicating if and how much progress should be printed during the computation of the tests for independence.
`idT.verbose`	logical, passed as `verbose` argument to `indepTestSim()`.
`piTest`	(d, d)-`matrix` of `indepTest` objects as returned by `pairwiseIndepTest()`.
`pvalues`	(d, d)-`matrix` of p-values.

method — character vector of adjustment methods for p-values; see p.adjust.methods for more details.

globalFun — function determining how to compute a global p-value from a matrix of pairwise adjusted p-values.

Value

pairwiseCcop (n, d, d)-array cu.u with cu.u[i,j] containing $C(u_i \,|\, u_j)$ for $i \neq j$ and u_i for $i = j$.

pairwiseIndepTest (d, d)-matrix of lists with test results as returned by indepTest(). The test results correspond to pairwise tests of independence as conducted by indepTest().

pviTest (d, d)-matrix of p-values.

gpviTest global p-values for the specified methods.

Note

If u are distributed according to or “perfectly” sampled from a copula,)Note that (typically) pseudo-observations or perfectly simulated

References

Hofert and Mächler (2013), see pairsRosenblatt.

See Also

pairsRosenblatt for where these tools are used, including demo(gof_graph) for examples.

Examples

```
## demo(gof_graph)
```

---

gnacopula — *Goodness-of-fit Testing for (Nested) Archimedean Copulas*

Description

gnacopula() conducts a goodness-of-fit test for the given (H_0-)copula cop based on the (copula-)data u.

NOTE: gnacopula() is deprecated, call gofCopula() instead.

Usage

```
gnacopula(u, cop, n.bootstrap,
          estim.method = eval(formals(enacopula)$method),
          include.K=TRUE, n.MC=0, trafo=c("Hering.Hofert", "Rosenblatt"),
          method=eval(formals(gofTstat)$method), verbose=TRUE, ...)
```

Arguments

u	$n \times d$-matrix of values in $[0, 1]$; should be (pseudo-/copula-)observations from the copula to be tested. Consider applying the function pobs() first in order to obtain u.
cop	H_0-"outer_nacopula" with specified parameters to be tested for (currently only Archimedean copulas are provided).
n.bootstrap	positive integer specifying the number of bootstrap replicates.
estim.method	character string determining the estimation method; see enacopula(). We currently only recommend the default "mle" (or maybe "smle").
include.K	logical indicating whether the last component, involving the Kendall distribution function K(), is used in the transformation htrafo() of Hering and Hofert (2011). Note that this only applies to trafo="Hering.Hofert".
n.MC	parameter n.MC for htrafo() (and thus for K()) if trafo="Hering.Hofert" and for cCopula() if trafo="Rosenblatt".
trafo	a character string specifying the multivariate transformation performed for goodness-of-fit testing, which has to be one (or a unique abbreviation) of "Hering.Hofert" for the multivariate transformation of Hering and Hofert (2011); see htrafo(). "Rosenblatt" for the multivariate transformation of Rosenblatt (1952); see cCopula().
method	a character string specifying the goodness-of-fit test statistic to be used; see gofTstat().
verbose	if TRUE, the progress of the bootstrap is displayed via txtProgressBar.
...	additional arguments passed to enacopula().

Details

The function gnacopula() performs a parametric bootstrap for the goodness-of-fit test specified by trafo and method. The transformation given by trafo specifies the multivariate transformation which is first applied to the (copula-) data u (typically, the pseudo-observations are used); see htrafo() or cCopula() for more details. The argument method specifies the particular goodness-of-fit test carried out, which is either the Anderson-Darling test for the univariate standard uniform distribution (for method="AnChisq" or method="AnGamma") in a one-dimensional setup or the tests described in Genest et al. (2009) for the multivariate standard uniform distribution directly in a multivariate setup. As estimation method, the method provided by estim.method is used.

Note that a finite-sample correction is made when computing p-values; see gofCopula() for details.

A word of warning: Do work carefully with the variety of different goodness-of-fit tests that can be performed with gnacopula(). For example, among the possible estimation methods at hand, only MLE is known to be consistent (under conditions to be verified). Furthermore, for the tests based on the Anderson-Darling test statistic, it is theoretically not clear whether the parametric bootstrap converges. Consequently, the results obtained should be treated with care. Moreover, several estimation methods are known to be prone to numerical errors (see Hofert et al. (2013)) and are thus not recommended to be used in the parametric bootstrap. A warning is given if gnacopula() is called with a method not being MLE.

Value

gnacopula returns an R object of class "htest". This object contains a list with the bootstrap results including the components

p.value: the bootstrapped p-value;

statistic: the value of the test statistic computed for the data u;

data.name: the name of u;

method: a character describing the goodness-of-fit test applied;

estimator: the estimator computed for the data u;

bootStats: a list with component estimator containing the estimators for all bootstrap replications and component statistic containing the values of the test statistic for each bootstrap replication.

References

Genest, C., Rémillard, B., and Beaudoin, D. (2009), Goodness-of-fit tests for copulas: A review and a power study *Insurance: Mathematics and Economics* **44**, 199–213.

Rosenblatt, M. (1952), Remarks on a Multivariate Transformation, *The Annals of Mathematical Statistics* **23**, 3, 470–472.

Hering, C. and Hofert, M. (2011), Goodness-of-fit tests for Archimedean copulas in large dimensions, submitted.

Hofert, M., Mächler, M., and McNeil, A. J. (2012). Likelihood inference for Archimedean copulas in high dimensions under known margins. *Journal of Multivariate Analysis* **110**, 133–150.

See Also

gofTstat() for the implemented test statistis, htrafo() and cCopula() involved and K() for the Kendall distribution function.

gofCopula() for other (parametric bootstrap) based goodness-of-fit tests.

---

gofCopula   *Goodness-of-fit Tests for Copulas*

Description

The goodness-of-fit tests are based, by default, on the empirical process comparing the empirical copula with a parametric estimate of the copula derived under the null hypothesis, the default test statistic, "Sn", being the Cramer-von Mises functional S_n defined in Equation (2) of Genest, Remillard and Beaudoin (2009). In that case, approximate p-values for the test statistic can be obtained either using a *parametric bootstrap* (see references two and three) or by means of a faster *multiplier* approach (see references four and five).

Alternative test statistics can be used, in particular if a *parametric bootstrap* is employed.

The prinicipal function is gofCopula() which, depending on simulation either calls gofPB() or gofMB().

Usage

```
## Generic [and "rotCopula" method] ------ Main function ------
gofCopula(copula, x, ...)
## S4 method for signature 'copula'
gofCopula(copula, x, N = 1000,
          method = c("Sn", "SnB", "SnC", "Rn"),
          estim.method = c("mpl", "ml", "itau", "irho", "itau.mpl"),
          simulation = c("pb", "mult"), verbose = interactive(), ties = NA,
          ties.method = c("max", "average", "first", "last", "random", "min"),
          fit.ties.meth = eval(formals(rank)$ties.method), ...)

## Internal 'helper' functions : ---
gofPB(copula, x, N, method = c("Sn", "SnB", "SnC"),
      estim.method = c("mpl", "ml", "itau", "irho", "itau.mpl"),
      trafo.method = if(method == "Sn") "none" else c("cCopula", "htrafo"),
      trafoArgs = list(), verbose = interactive(), useR = FALSE, ties = NA,
      ties.method = c("max", "average", "first", "last", "random", "min"),
      fit.ties.meth = eval(formals(rank)$ties.method), ...)

gofMB(copula, x, N, method = c("Sn", "Rn"),
      estim.method = c("mpl", "ml", "itau", "irho"),
      verbose = interactive(), useR = FALSE, m = 1/2, zeta.m = 0,
      b = 1/sqrt(nrow(x)),
      ties.method = c("max", "average", "first", "last", "random", "min"),
      fit.ties.meth = eval(formals(rank)$ties.method), ...)
```

Arguments

`copula`	object of class "`copula`" representing the hypothesized copula family.
`x`	a data matrix that will be transformed to pseudo-observations using `pobs()`.
`N`	number of bootstrap or multiplier replications to be used to obtain approximate realizations of the test statistic under the null hypothesis.
`method`	a `character` string specifying the goodness-of-fit test statistic to be used. For `simulation = "pb"`, one of "Sn", "SnB" or "SnC" with `trafo.method != "none"` if `method != "Sn"`. For `simulation = "mult"`, one of `"Sn"` or `"Rn"`, where the latter is R_n from Genest et al. (2013).
`estim.method`	a character string specifying the estimation method to be used to estimate the dependence parameter(s); see `fitCopula()`.
`simulation`	a string specifying the resampling method for generating approximate realizations of the test statistic under the null hypothesis; can be either `"pb"` (parametric bootstrap) or `"mult"` (multiplier).
`verbose`	a logical specifying if progress of the parametric bootstrap should be displayed via `txtProgressBar`.
`...`	for `gofCopula`, additional arguments passed to `gofPB()` or `gofMB()`; for `gofPB()` and `gofMB()`: additional arguments passed to `fitCopula()`. These may notably contain `hideWarnings`, and `optim.method`, `optim.control`, `lower`, or `upper` depending on the `optim.method`.

`trafo.method`	only for the parametric bootstrap (`"pb"`): String specifying the transformation to $U[0,1]^d$; either `"none"` or one of `"cCopula"`, see `cCopula()`, or `"htrafo"`, see `htrafo()`. If `method != "Sn"`, one needs to set `trafo.method != "none"`.
`trafoArgs`	only for the parametric bootstrap. A `list` of optional arguments passed to the transformation method (see `trafo.method` above).
`useR`	logical indicating whether an R or C implementation is used.
`ties.method`	string specifying how ranks should be computed, except for fitting, if there are ties in any of the coordinate samples of `x`; passed to `pobs`.
`fit.ties.meth`	string specifying how ranks should be computed when fitting by maximum pseudo-likelihood if there are ties in any of the coordinate samples of `x`; passed to `pobs`.
`ties`	only for the parametric bootstrap. Logical indicating whether a version of the parametric boostrap adapted to the presence of ties in any of the coordinate samples of `x` should be used; the default value of `NA` indicates that the presence/absence of ties will be checked for automatically.
`m, zeta.m`	only for the multiplier with `method = "Rn"`. `m` is the power and `zeta.m` is the adjustment parameter ζ_m for the denominator of the test statistic.
`b`	only for the multiplier. `b` is the bandwidth required for the estimation of the first-order partial derivatives based on the empirical copula.

Details

If the parametric bootstrap is used, the dependence parameters of the hypothesized copula family can be estimated by any estimation method available for the family, up to a few exceptions. If the multiplier is used, any of the rank-based methods can be used in the bivariate case, but only maximum pseudo-likelihood estimation can be used in the multivariate (multiparameter) case.

The price to pay for the higher computational efficiency of the multiplier is more programming work as certain partial derivatives need to be computed for each hypothesized parametric copula family. When estimation is based on maximization of the pseudo-likelihood, these have been implemented for six copula families thus far: the Clayton, Gumbel-Hougaard, Frank, Plackett, normal and t copula families. For other families, numerical differentiation based on `grad()` from package **numDeriv** is used (and a warning message is displayed).

Although the empirical processes involved in the multiplier and the parametric bootstrap-based test are asymptotically equivalent under the null, the finite-sample behavior of the two tests might differ significantly.

Both for the parametric bootstrap and the multiplier, the approximate p-value is computed as

$$(0.5 + \sum_{b=1}^{N} \mathbf{1}_{\{T_b \geq T\}})/(N+1),$$

where T and T_b denote the test statistic and the bootstrapped test statistc, respectively. This ensures that the approximate p-value is a number strictly between 0 and 1, which is sometimes necessary for further treatments. See Pesarin (2001) for more details.

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.

The former argument print.every is deprecated and not supported anymore; use verbose instead.

Value

An object of class htest which is a list, some of the components of which are

statistic	value of the test statistic.
p.value	corresponding approximate p-value.
parameter	estimates of the parameters for the hypothesized copula family.

Note

These tests were theoretically studied and implemented under the assumption of continuous margins, which implies that ties in the component samples occur with probability zero. The presence of ties in the data might substantially affect the approximate p-values. Through argument ties, the user can however select a version of the parametric bootstrap adapted to the presence of ties. No such adaption exists for the multiplier for the moment.

References

Genest, C., Huang, W., and Dufour, J.-M. (2013). A regularized goodness-of-fit test for copulas. *Journal de la Société française de statistique* **154**, 64–77.

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

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

Kojadinovic, I., Yan, J., and Holmes M. (2011). Fast large-sample goodness-of-fit tests for copulas. *Statistica Sinica* **21**, 841–871.

Kojadinovic, I. and Yan, J. (2011). A goodness-of-fit test for multivariate multiparameter copulas based on multiplier central limit theorems. *Statistics and Computing* **21**, 17–30.

Kojadinovic, I. and Yan, J. (2010). Modeling Multivariate Distributions with Continuous Margins Using the copula R Package. *Journal of Statistical Software* **34**(9), 1–20, http://www.jstatsoft.org/v34/i09/.

Kojadinovic, I. (2017). Some copula inference procedures adapted to the presence of ties. *Computational Statistics and Data Analysis* **112**, 24–41, http://arxiv.org/abs/1609.05519.

Pesarin, F. (2001). *Multivariate Permutation Tests: With Applications in Biostatistics*. Wiley.

See Also

fitCopula() for the underlying estimation procedure and gofTstat() for details on *some* of the available test statistics.

Examples

```
## The following example is available in batch through
## demo(gofCopula)

n <- 200; N <- 1000 # realistic (but too large for interactive use)
n <-  60; N <-  200 # (time (and tree !) saving ...)

## A two-dimensional data example ----------------------------------
x <- rCopula(n, claytonCopula(3))

## Does the Gumbel family seem to be a good choice (statistic "Sn")?
gofCopula(gumbelCopula(), x, N=N)
## With "SnC", really s..l..o..w.. --- with "SnB", *EVEN* slower
gofCopula(gumbelCopula(), x, N=N, method = "SnC", trafo.method = "cCopula")
## What about the Clayton family?
gofCopula(claytonCopula(), x, N=N)

## Similar with a different estimation method
gofCopula(gumbelCopula (), x, N=N, estim.method="itau")
gofCopula(claytonCopula(), x, N=N, estim.method="itau")


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

## Does the Gumbel family seem to be a good choice?
g.copula <- gumbelCopula(dim = 3)
gofCopula(g.copula, x, N=N)
## What about the t copula?
t.copula <- tCopula(dim = 3, dispstr = "un", df.fixed = TRUE)
if(FALSE) ## this is *VERY* slow currently
  gofCopula(t.copula, x, N=N)

## The same with a different estimation method
gofCopula(g.copula, x, N=N, estim.method="itau")
if(FALSE) # still really slow
  gofCopula(t.copula, x, N=N, estim.method="itau")

## The same using the multiplier approach
gofCopula(g.copula, x, N=N, simulation="mult")
gofCopula(t.copula, x, N=N, simulation="mult")
if(FALSE) # no yet possible
    gofCopula(t.copula, x, N=N, simulation="mult", estim.method="itau")
```

---

gofEVCopula   *Goodness-of-fit Tests for Bivariate Extreme-Value Copulas*

Description

Goodness-of-fit tests for extreme-value copulas based on the empirical process comparing one of the two nonparameteric rank-based estimator of the Pickands dependence function studied in Genest and Segers (2009) with a parametric estimate of the Pickands dependence function derived under the null hypothesis. The test statistic is the Cramer-von Mises functional Sn defined in Equation (5) of Genest, Kojadinovic, G. Nešlehová, and Yan (2010). Approximate p-values for the test statistic are obtained using a parametric bootstrap.

Usage

```
gofEVCopula(copula, x, N = 1000,
            method = c("mpl", "ml", "itau", "irho"),
            estimator = c("CFG", "Pickands"), m = 1000,
            verbose = interactive(),
           ties.method = c("max", "average", "first", "last", "random", "min"),
            fit.ties.meth = eval(formals(rank)$ties.method), ...)
```

Arguments

`copula`	object of class "`evCopula`" representing the hypothesized extreme-value copula family.
`x`	a data matrix that will be transformed to pseudo-observations.
`N`	number of bootstrap samples to be used to simulate realizations of the test statistic under the null hypothesis.
`method`	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).
`estimator`	specifies which nonparametric rank-based estimator of the unknown Pickands dependence function to use; can be either `"CFG"` (Caperaa-Fougeres-Genest) or `"Pickands"`.
`m`	number of points of the uniform grid on [0,1] used to compute the test statistic numerically.
`verbose`	a logical specifying if progress of the bootstrap should be displayed via `txtProgressBar`.
`ties.method`	string specifying how ranks should be computed, except for fitting, if there are ties in any of the coordinate samples of `x`; passed to `pobs`.
`fit.ties.meth`	string specifying how ranks should be computed when fitting by maximum pseudo-likelihood if there are ties in any of the coordinate samples of `x`; passed to `pobs`.
`...`	further optional arguments, passed to `fitCopula()`, notably `optim.method`, the `method` for `optim()`. In **copula** versions `0.999-14` and earlier, the default for that was `"Nelder-Mead"`, but now is the same as for `fitCopula()`.

Details

More details can be found in the second reference.

The former argument `print.every` is deprecated and not supported anymore; use `verbose` instead.

Value

An object of class htest which is a list, some of the components of which are

statistic	value of the test statistic.
p.value	corresponding approximate p-value.
parameter	estimates of the parameters for the hypothesized copula family.

Note

For a given degree of dependence, the most popular bivariate extreme-value copulas are strikingly similar.

References

Genest, C. and Segers, J. (2009). Rank-based inference for bivariate extreme-value copulas. *Annals of Statistics* **37**, 2990–3022.

Genest, C. Kojadinovic, I., G. Nešlehová, J., and Yan, J. (2011). A goodness-of-fit test for bivariate extreme-value copulas. *Bernoulli* **17**(1), 253–275.

See Also

evCopula, evTestC, evTestA, evTestK, gofCopula, An.

Examples

```
n <- 100; N <- 1000 # realistic (but too large currently for CRAN checks)
n <-  60; N <-  200 # (time (and tree !) saving ...)
x <- rCopula(n, claytonCopula(3))


## Does the Gumbel family seem to be a good choice?
gofEVCopula(gumbelCopula(), x, N=N)


## The same with different (and cheaper) estimation methods:
gofEVCopula(gumbelCopula(), x, N=N, method="itau")
gofEVCopula(gumbelCopula(), x, N=N, method="irho")


## The same with different extreme-value copulas
gofEVCopula(galambosCopula(), x, N=N)
gofEVCopula(galambosCopula(), x, N=N, method="itau")
gofEVCopula(galambosCopula(), x, N=N, method="irho")

gofEVCopula(huslerReissCopula(), x, N=N)
gofEVCopula(huslerReissCopula(), x, N=N, method="itau")
gofEVCopula(huslerReissCopula(), x, N=N, method="irho")

gofEVCopula(tevCopula(df.fixed=TRUE), x, N=N)
gofEVCopula(tevCopula(df.fixed=TRUE), x, N=N, method="itau")
```

```
gofEVCopula(tevCopula(df.fixed=TRUE), x, N=N, method="irho")
```

---

gofOtherTstat *Various Goodness-of-fit Test Statistics*

Description

`gofBTstat()` computes supposedly Beta distributed test statistics for checking uniformity of `u` on the unit sphere.

Usage

```
gofBTstat(u)
```

Arguments

`u`	(n, d)-matrix of values whose rows supposedly follow a uniform distribution on the unit sphere in $\mathbf{R}^d$.

Value

An $(n, d-1)$-`matrix` where the (i, k)th entry is

$$B_{ik} = \frac{\sum_{j=1}^{k} u_{ij}^2}{\sum_{j=1}^{d} u_{ij}^2}.$$

References

Li, R.-Z., Fang, K.-T., and Zhu, L.-X. (1997). Some Q-Q probability plots to test spherical and elliptical symmetry. *Journal of Computational and Graphical Statistics* **6**(4), 435–450.

Examples

```
## generate data on the unit sphere
n <- 360
d <- 5
set.seed(1)
x <- matrix(rnorm(n*d), ncol=d)
U <- x/sqrt(rowSums(x^2))

## compute the test statistics B_k, k in {1,..,d-1}
Bmat <- gofBTstat(U)

## (graphically) check if Bmat[,k] follows a Beta(k/2, (d-k)/2) distribution
qqp <- function(k, Bmat) {
    d <- ncol(Bmat)+1L
    tit <- substitute(plain("Beta")(.A.,.B.)~~ bold("Q-Q Plot"),
                      list(.A. = k/2, .B. = (d-k)/2))
```

```
    qqplot2(Bmat[,k], qF=function(p) qbeta(p, shape1=k/2, shape2=(d-k)/2),
            main.args=list(text=tit, side=3, cex=1.3, line=1.1, xpd=NA))
}
qqp(1, Bmat=Bmat) # k=1
qqp(3, Bmat=Bmat) # k=3
```

---

gofTstat *Goodness-of-fit Test Statistics*

Description

gofTstat() computes various goodness-of-fit test statistics typically used in gofCopula(*, simulation = "pb").

Usage

```
gofTstat(u, method = c("Sn", "SnB", "SnC", "AnChisq", "AnGamma"),
         useR = FALSE, ...)
```

Arguments

u — $n \times d$-matrix of values in $[0, 1]$, supposedly independent uniform observations in the hypercube, that is, $U_i \sim U[0, 1]^d$, i.i.d., for $i \in \{1, \ldots, n\}$.

method — a character string specifying the goodness-of-fit test statistic to be used, which has to be one (or a unique abbreviation) of

"Sn" for computing the test statistic S_n from Genest, Rémillard, Beaudoin (2009).

"SnB" for computing the test statistic $S_n^{(B)}$ from Genest, Rémillard, Beaudoin (2009).

"SnC" for computing the test statistic $S_n^{(C)}$ from Genest et al. (2009).

"AnChisq" Anderson-Darling test statistic for computing (supposedly) U$[0, 1]$-distributed (under H_0) random variates via the distribution function of the chi-square distribution with d degrees of freedom. To be more precise, the Anderson-Darling test statistc of the variates

$$\chi_d^2\Big(\sum_{j=1}^{d}(\Phi^{-1}(u_{ij}))^2\Big)$$

is computed (via ADGofTest::ad.test), where Φ^{-1} denotes the quantile function of the standard normal distribution function, χ_d^2 denotes the distribution function of the chi-square distribution with d degrees of freedom, and u_{ij} is the jth component in the ith row of u.

"AnGamma" similar to method="AnChisq" but based on the variates

$$\Gamma_d\Big(\sum_{j=1}^{d}(-\log u_{ij})\Big),$$

where Γ_d denotes the distribution function of the gamma distribution with shape parameter d and shape parameter one (being equal to an Erlang(d) distribution function).

`useR` logical indicating whether an R or C implementation is used.

`...` additional arguments passed for computing the different test statistics.

Details

This function should be used with care. The different test statistics were implemented (partly) for different purposes and goodness-of-fit tests and should be used only with knowledge about such (see the references for more details).

Value

The value of the test statistic, a `numeric`.

References

Genest, C., Rémillard, B., and Beaudoin, D. (2009), Goodness-of-fit tests for copulas: A review and a power study *Insurance: Mathematics and Economics* **44**, 199–213.

Rosenblatt, M. (1952), Remarks on a Multivariate Transformation, *The Annals of Mathematical Statistics* **23**, 3, 470–472.

Hering, C. and Hofert, M. (2014), Goodness-of-fit tests for Archimedean copulas in high dimensions, *Innovations in Quantitative Risk Management*.

Hofert, M., Mächler, M., and McNeil, A. J. (2012). Likelihood inference for Archimedean copulas in high dimensions under known margins. *Journal of Multivariate Analysis* **110**, 133–150.

See Also

`gofCopula()` for goodness-of-fit tests where (some of) these test statistics are used.

Examples

```
## generate data
cop <- archmCopula("Gumbel", param=iTau(gumbelCopula(), 0.5), dim=5)
set.seed(1)
U <- rCopula(1000, cop)

## compute Sn (as is done in a parametric bootstrap, for example)
Uhat <- pobs(U) # pseudo-observations
u <- cCopula(Uhat, copula = cop) # Rosenblatt transformed data (with correct copula)
gofTstat(u, method = "Sn", copula = cop) # compute test statistic Sn; requires copula argument
```

htrafo *GOF Testing Transformation of Hering and Hofert*

Description

The transformation described in Hering and Hofert (2014), for Archimedean copulas.

Usage

```
htrafo(u, copula, include.K = TRUE, n.MC = 0, inverse = FALSE,
       method = eval(formals(qK)$method), u.grid, ...)
```

Arguments

u	$n \times d$-matrix with values in $[0, 1]$. If inverse=FALSE (the default), u contains (pseudo-/copula-)observations from the copula copula based on which the transformation is carried out; consider applying the function pobs() first in order to obtain u. If inverse=TRUE, u contains $U[0, 1]^d$ distributed values which are transformed to copula-based (copula) ones.
copula	an Archimedean copula specified as "outer_nacopula" or "archmCopula".
include.K	logical indicating whether the last component, involving the Kendall distribution function K, is used in htrafo().
n.MC	parameter n.MC for K.
inverse	logical indicating whether the inverse of the transformation is returned.
method	method to compute qK().
u.grid	argument of qK() (for method="discrete").
...	additional arguments passed to qK() if inverse = TRUE.

Details

Given a d-dimensional random vector $\boldsymbol{U}$ following an Archimedean copula C with generator ψ, Hering and Hofert (2014) showed that $\boldsymbol{U}' \sim \mathrm{U}[0, 1]^d$, where

$$U'_j = \left(\frac{\sum_{k=1}^{j} \psi^{-1}(U_k)}{\sum_{k=1}^{j+1} \psi^{-1}(U_k)} \right)^j, \; j \in \{1, \dots, d-1\}, \; U'_d = K(C(\boldsymbol{U})).$$

htrafo applies this transformation row-wise to u and thus returns either an $n \times d$- or an $n \times (d-1)$-matrix, depending on whether the last component U'_d which involves the (possibly numerically challenging) Kendall distribution function K is used (include.K=TRUE) or not (include.K=FALSE).

Value

htrafo() returns an $n \times d$- or $n \times (d-1)$-matrix (depending on whether include.K is TRUE or FALSE) containing the transformed input u.

References

Hering, C. and Hofert, M. (2014). Goodness-of-fit tests for Archimedean copulas in high dimensions. *Innovations in Quantitative Risk Management*.

Examples

```
## Sample and build pseudo-observations (what we normally have available)
## of a Clayton copula
tau <- 0.5
theta <- iTau(claytonCopula(), tau = tau)
d <- 5
cc <- claytonCopula(theta, dim = d)
set.seed(271)
n <- 1000
U <- rCopula(n, copula = cc)
X <- qnorm(U) # X now follows a meta-Gumbel model with N(0,1) marginals
U <- pobs(X) # build pseudo-observations

## Graphically check if the data comes from a meta-Clayton model
## with the transformation of Hering and Hofert (2014):
U.H <- htrafo(U, copula = cc) # transform the data
splom2(U.H, cex = 0.2) # looks good

## The same for an 'outer_nacopula' object
cc. <- onacopulaL("Clayton", list(theta, 1:d))
U.H. <- htrafo(U, copula = cc.)
splom2(U.H., cex = 0.2) # looks good

## What about a meta-Gumbel model?
## The parameter is chosen such that Kendall's tau equals (the same) tau
gc <- gumbelCopula(iTau(gumbelCopula(), tau = tau), dim = d)

## Plot of the transformed data (Hering and Hofert (2014)) to see the
## deviations from uniformity
U.H.. <- htrafo(U, copula = gc)
splom2(U.H.., cex = 0.2) # deviations visible
```

---

indepCopula *Construction of Independence Copula Class Objects*

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(10, indep.cop)
dCopula(x, indep.cop)
persp(indepCopula(), pCopula)
```

---

indepCopula-class *Class "indepCopula"*

Description

Independence 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

`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.

`fullname`**:** **deprecated**; object of class `"character"`, family names of the copula.

Methods

A `signature(copula = "indepCopula")`: ...

dCopula `signature(copula = "indepCopula")`: ...

pCopula `signature(copula = "indepCopula")`: ...

rCopula `signature(copula = "indepCopula")`: ...

Extends

Class `"indepCopula"` extends classes `"archmCopula"` and `"evCopula"` directly.

See Also

`indepCopula`, `copula-class`.

Examples

```
getClass("indepCopula")
```

---

indepTest — *Test Independence of Continuous Random Variables via Empirical Copula*

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, verbose = interactive())
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.
verbose	a logical specifying if progress should be displayed via txtProgressBar.
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 returned by 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

The current (C code) implementation of indepTestSim() uses (RAM) memory of size $O(n^2p)$, and time

$O(Nn^2p)$. This renders it unfeasible when n is large.

See the references below for more details, especially Genest and Rémillard (2004).

The former argument print.every is deprecated and not supported anymore; use verbose instead.

Value

The function indepTestSim() returns an object of class "indepTestDist" whose attributes are: sample.size, data.dimension, max.card.subsets, number.repetitons, 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 Genest *et al* (2007), 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 Genest *et al* (2007), p.175) and global.statistic.pvalue (corresponding p-value).

References

Deheuvels, P. (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.

Deheuvels, P. (1981) A non parametric test for independence, *Publ. Inst. Statist. Univ. Paris*. **26**, 29–50.

Genest, C. and Rémillard, B. (2004) Tests of independence and randomness based on the empirical copula process. *Test* **13**, 335–369.

Genest, C., Quessy, J.-F., and Rémillard, B. (2006). Local efficiency of a Cramer-von Mises test of independence, *Journal of Multivariate Analysis* **97**, 274–294.

Genest, C., Quessy, J.-F., and Rémillard, B. (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:

set.seed(2004)
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.

## For a realistic N = 1000 (default), this takes a few seconds:
N. <- if(copula:::doExtras()) 1000 else 120
N.
system.time(d <- indepTestSim(100, 5, N = N.))
## For N=1000,  2 seconds (lynne 2015)
## You could save 'd' for future use, via  saveRDS()

## The next step consists of performing the test itself (and print its results):
(iTst <- indepTest(x,d))

## Display the dependogram with the details:
dependogram(iTst, 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:
system.time( d <- indepTestSim(100,5,3, N=N.) )
iTy <- indepTest(y,d)
iTy
dependogram(iTy, print=TRUE)
```

`initOpt` *Initial Interval or Value for Parameter Estimation of Archimedean Copulas*

Description

Compute an initial interval or initial value for optimization/estimation routines (only a heuristic; if this fails, choose your own interval or value).

Usage

```
initOpt(family, tau.range=NULL, interval = TRUE, u,
        method = c("tau.Gumbel", "tau.mean"), warn = TRUE, ...)
```

Arguments

`family`	Archimedean family to find an initial interval for.
`tau.range`	numeric vector containing lower and upper admissible Kendall's tau, or `NULL` which choses family-specific defaults, see the function definition.
`interval`	`logical` indicating whether an initial interval (the default) or an initial value should be returned.
`u`	matrix of realizations following the copula family specified by `family`. Note that `u` can be omitted if `interval=TRUE`.
`method`	a `character` string specifying the method to be used to compute an estimate of Kendall's tau. This has to be one (or a unique abbreviation) of `"tau.Gumbel"` an estimator based on the diagonal maximum-likelihood estimator for Gumbel is used. `"tau.mean"` an estimator based on the mean of pairwise sample versions of Kendall's tau is applied.
`warn`	logical indicating if warnings are printed for `method="tau.Gumbel"` when the diagonal maximum-likelihood estimator is smaller than 1.
`...`	additional arguments passed to `cor()` when `method="tau.mean"`. Note that otherwise (no additional arg.), the much faster `cor.fk()` from package **pcaPP** is used.

Details

For `method="tau.mean"` and `interval=FALSE`, the mean of pairwise sample versions of Kendall's tau is computed as an estimator of the Kendall's tau of the Archimedean copula family provided. This can be slow, especially if the dimension is large. Method `method="tau.Gumbel"` (the default) uses the explicit and thus very fast diagonal maximum-likelihood estimator for Gumbel's family to find initial values. Given this estimator $\hat{\theta}^{\mathrm{G}}$, the corresponding Kendall's tau is $\tau^{\mathrm{G}}(\hat{\theta}^{\mathrm{G}})$ where $\tau^{\mathrm{G}}(\theta) = (\theta - 1)/\theta$ denotes Kendall's tau for Gumbel. This provides an estimator of Kendall's tau which is typically much faster to evaluate than, pairwise Kendall's taus. Given the estimated 'amount of concordance' based on Kendall's tau, one can obtain an initial value for the provided family by applying τ^{-1}, that is, the inverse of Kendall's tau of the family for which the initial value is to be computed. Note that if the estimated Kendall's tau does not lie in the range of Kendall's tau as provided by the bivariate vector `tau.range`, the point in `tau.range` closest to the estimated Kendall's tau is chosen.

The default (`interval=TRUE`) returns a reasonably large initial interval; see the default of `tau.range` in the definition of `initOpt` for the chosen values (in terms of Kendall's tau). These default values cover a large range of concordance. If this interval is (still) too small, one can adjust it by providing `tau.range`. If it is too large, a 'distance to concordance' can be used to determine parameter values such that the corresponding Kendall's taus share a certain distance to the initial value. For more details, see Hofert et al. (2012). Finally, let us note that for the case `interval=TRUE`, `u` is not required.

Value

initial interval which can be used for optimization (for example, for `emle`).

References

Hofert, M., Mächler, M., and McNeil, A. J. (2012). Likelihood inference for Archimedean copulas in high dimensions under known margins. *Journal of Multivariate Analysis* **110**, 133–150.

See Also

`enacopula`, `emle`, `edmle`, `emde`, and `ebeta` (where `initOpt` is applied to find initial intervals).

Examples

```
## Definition of the function:
initOpt

## Generate some data:
tau <- 0.25
(theta <- copGumbel@iTau(tau)) # 4/3
d <- 20
(cop <- onacopulaL("Gumbel", list(theta,1:d)))

set.seed(1)
n <- 200
U <- rnacopula(n, cop)

## Initial interval:
initOpt("Gumbel") # contains theta

## Initial values:
initOpt("Gumbel", interval=FALSE, u=U) # 1.3195
initOpt("Gumbel", interval=FALSE, u=U, method="tau.mean") # 1.2844
```

interval *Construct Simple "interval" Object*

Description

Easy construction of an object of class interval, using typical mathematical notation.

Usage

```
interval(ch)
```

Arguments

ch	a character string specifying the interval.

Value

an interval object.

See Also

the interval class documentation, notably its reference to more sophisticated interval classes available for R.

Examples

```
interval("[0, 1)")

## Two ways to specify open interval borders:
identical(interval("]-1,1["),
          interval("(-1,1)"))

## infinite :
interval("[0, Inf)")

## arithmetic with scalars works:
4 + 2 * interval("[0, 1.5)") # ->  [4, 7)

## str() to look at internals:
str( interval("[1.2, 7]") )
```

`interval-class` *Class "interval" of Simple Intervals*

Description

The S4 `class` `"interval"` is a simple class for numeric intervals.

`"maybeInterval"` is a class union (see `setClassUnion`) of `"interval"` and `"NULL"`.

Objects from the Class

Objects can be created by calls of the form `new("interval", ...)`, but typically they are built via `interval()`.

Slots

`.Data`**:** numeric vector of length two, specifying the interval ranges.

`open`**:** `logical` vector of length two, specifying if the interval is open or closed on the left and right, respectively.

Extends

Class `"interval"` extends `"numeric"`, from data part, and `"maybeInterval"`, directly.

Methods

"%in%" `signature(x = "numeric", table = "interval")`: check if `x` is inside the interval, carefully differentiating open and closed intervals.

format `signature(x = "interval")`: ...

show `signature(object = "interval")`: ...

Summary `signature(x = "interval")`: Group methods, notably `range()`, `min()`, etc.

Note

There are more sophisticated interval classes, functions and methods, notably in package **Intervals**. We only use this as a simple interface in order to specify our copula functions consistently.

See Also

`interval` constructs "interval" objects conveniently.

Examples

```
-1:2 %in% interval("(0, Inf)")
## 0 is *not* inside
```

K *Kendall Distribution Function for Archimedean Copulas*

Description

The *Kendall distribution* of an Archimedean copula is defined by

$$K(u) = P(C(U_1, U_2, \dots, U_d) \le u),$$

where $u \in [0, 1]$, and the d-dimensional $(U_1, U_2, \dots, U_d)$ is distributed according to the copula C. Note that the random variable $C(U_1, U_2, \dots, U_d)$ is known as "probability integral transform". Its distribution function K is equal to the identity if $d = 1$, but is non-trivial for $d \ge 2$.

`Kn()` computes the empirical Kendall distribution function, `pK()` the distribution function (so $K()$ itself), `qK()` the quantile function, `dK()` the density, and `rK()` random number generation from $K()$ for an Archimedean copula.

Usage

```
Kn(u, x) # empirical Kendall distribution function
dK(u, copula, d, n.MC = 0, log.p = FALSE) # density
pK(u, copula, d, n.MC = 0, log.p = FALSE) # df
qK(p, copula, d, n.MC = 0, log.p = FALSE, # quantile function
   method = c("default", "simple", "sort", "discrete", "monoH.FC"),
   u.grid, ...)
rK(n, copula, d) # random number generation
```

Arguments

`u`	evaluation point(s) (in $[0, 1]$).
`x`	data (in the d-dimensional space) based on which the Kendall distribution function is estimated.
`copula`	`acopula` with specified parameter, or (currently for `rK` only) a `outer_nacopula`.
`d`	dimension (not used when `copula` is an `outer_nacopula`).
`n.MC`	`integer`, if positive, a Monte Carlo approach is applied with sample size equal to `n.MC` to evaluate the generator derivatives involved; otherwise (`n.MC = 0`) the exact formula is used based on the generator derivatives as found by Hofert et al. (2012).
`log.p`	`logical`; if `TRUE`, probabilities p are given as $\log p$.
`p`	probabilities or log-probabilities if `log.p` is true.
`method`	string for the method to compute the quantile function of K. Currently, one of **"default"** default method. Currently chooses `method="monoH.FC"` with `u.grid = 0:128/128`. This is fast but not too accurate (see example). **"simple"** straightforward root finding based on `uniroot`.

"sort" root finding based on uniroot but first sorting u.

"discrete" first, K is evaluated at the given grid points u.grid (which should contain 0 and 1). Based on these probabilities, quantiles are computed with findInterval.

"monoH.FC" first, K is evaluated at the given grid points u.grid. A monotone spline is then used to approximate K. Based on this approximation, quantiles are computed with uniroot.

u.grid (for method="discrete":) The grid on which K is evaluated, a numeric vector.

... additional arguments passed to uniroot (for method="default", method="simple", method="sort", and method="monoH.FC") or findInterval (for method="discrete"), notably tol (uniroot) for increased accuracy.

n sample size for rK.

Details

For a completely monotone Archimedean generator ψ,

$$K(u) = \sum_{k=0}^{d-1} \frac{\psi^{(k)}(\psi^{-1}(u))}{k!}(-\psi^{-1}(u))^k, \ u \in [0,1];$$

see Barbe et al. (1996). The corresponding density is

$$\frac{(-1)^d \psi^{(d)}(\psi^{-1}(u))}{(d-1)!}(-(\psi^{-1})'(u))(\psi^{-1}(u))^{d-1}$$

The empirical Kendall distribution function is computed as in Genest, G. Nešlehová, Ziegel (2011).

Value

The empirical Kendall distribution function, density, distribution function, quantile function and random number generator.

Note

Currently, the "default" method of qK() is fast but not very accurate, see the 'Examples' for more accuracy (with more CPU effort).

References

Barbe, P., Genest, C., Ghoudi, K., and Rémillard, B. (1996), On Kendall's Process, *Journal of Multivariate Analysis* **58**, 197–229.

Hofert, M., Mächler, M., and McNeil, A. J. (2012). Likelihood inference for Archimedean copulas in high dimensions under known margins. *Journal of Multivariate Analysis* **110**, 133–150.

Genest, C., G. Nešlehová, J., and Ziegel, J. (2011). Inference in multivariate Archimedean copula models. *TEST* **20**, 223–256.

See Also

htrafo or emde (where K is used); splinefun(*, "monoHC") for that method.

Examples

```
tau <- 0.5
(theta <- copGumbel@iTau(tau)) # 2
d <- 20
(cop <- onacopulaL("Gumbel", list(theta,1:d)))

## Basic check of the empirical Kendall distribution function
set.seed(271)
n <- 1000
U <- rCopula(n, copula = cop)
X <- qnorm(U)
K.sample <- pCopula(U, copula = cop)
u <- seq(0, 1, length.out = 256)
edfK <- ecdf(K.sample)
plot(u, edfK(u), type = "l", ylim = 0:1,
     xlab = quote(italic(u)), ylab = quote(K[n](italic(u)))) # simulated
K.n <- Kn(u, x = X)
lines(u, K.n, col = "royalblue3") # Kn
## Difference at 0
edfK(0) # edf of K at 0
K.n[1] # K_n(0); this is > 0 since K.n is the edf of a discrete distribution
## => therefore, Kn(K.sample, x=X) is not uniform
plot(Kn(K.sample, x = X), ylim = 0:1)
## Note: Kn(0) -> 0 for n -> Inf

## Compute Kendall distribution function
u <- seq(0,1, length.out = 255)
Ku    <- pK(u, copula = cop@copula, d = d) # exact
Ku.MC <- pK(u, copula = cop@copula, d = d, n.MC = 1000) # via Monte Carlo
stopifnot(all.equal(log(Ku),
    pK(u, copula = cop@copula, d = d, log.p=TRUE)))# rel.err 3.2e-16

## Build sample from K
set.seed(1)
n <- 200
W <- rK(n, copula = cop)

## Plot empirical distribution function based on W
## and the corresponding theoretical Kendall distribution function
## (exact and via Monte Carlo)
plot(ecdf(W), col = "blue", xlim = 0:1, verticals=TRUE,
     main = quote("Empirical"~ F[n](C(U)) ~
                    "and its Kendall distribution" ~ K(u)),
     do.points = FALSE, asp = 1)
abline(0,1, lty = 2); abline(h = 0:1, v = 0:1, lty = 3, col = "gray")
lines(u, Ku.MC, col = "red") # not quite monotone
lines(u, Ku, col = "black")  # strictly  monotone:
stopifnot(diff(Ku) >= 0)
legend(.25, .75, expression(F[n], K[MC](u), K(u)),
       col=c("blue" , "red", "black"), lty = 1, lwd = 1.5, bty = "n")

if(require("Rmpfr")) { # pK() now also works with high precision numbers:
```

```
 uM <- mpfr(0:255, 99)/256
 if(FALSE) {
   # not yet, now fails in  polyG() :
   KuM <- pK(uM, copula = cop@copula, d = d)
  ##  debug(copula:::.pK)
  debug(copula:::polyG)
 }
}# if( Rmpfr )


## Testing qK
pexpr <- quote( 0:63/63 );  p <- eval(pexpr)
d <- 10
cop <- onacopulaL("Gumbel", list(theta = 2, 1:d))
system.time(qK0 <- qK(p, copula = cop@copula, d = d)) # "default" - fast


system.time(qK1  <- qK(p, copula= cop@copula, d=d, method = "simple"))
system.time(qK1. <- qK(p, copula= cop@copula, d=d, method = "simple", tol = 1e-12))
system.time(qK2  <- qK(p, copula= cop@copula, d=d, method = "sort"))
system.time(qK2. <- qK(p, copula= cop@copula, d=d, method = "sort",   tol = 1e-12))
system.time(qK3  <- qK(p, copula= cop@copula, d=d, method = "discrete", u.grid = 0:1e4/1e4))
system.time(qK4  <- qK(p, copula= cop@copula, d=d, method = "monoH.FC",
                       u.grid = 0:5e2/5e2))
system.time(qK4. <- qK(p, copula= cop@copula, d=d, method = "monoH.FC",
                       u.grid = 0:5e2/5e2, tol = 1e-12))
system.time(qK5  <- qK(p, copula= cop@copula, d=d, method = "monoH.FC",
                       u.grid = 0:5e3/5e3))
system.time(qK5. <- qK(p, copula= cop@copula, d=d, method = "monoH.FC",
                       u.grid = 0:5e3/5e3, tol = 1e-12))
system.time(qK6  <- qK(p, copula= cop@copula, d=d, method = "monoH.FC",
                       u.grid = (0:5e3/5e3)^2))
system.time(qK6. <- qK(p, copula= cop@copula, d=d, method = "monoH.FC",
                       u.grid = (0:5e3/5e3)^2, tol = 1e-12))

## Visually they all coincide :
cols <- adjustcolor(c("gray50", "gray80", "light blue",
                      "royal blue", "purple3", "purple4", "purple"), 0.6)
matplot(p, cbind(qK0, qK1, qK2, qK3, qK4, qK5, qK6), type = "l", lwd = 2*7:1, lty = 1:7, col = cols,
        xlab = bquote(p == .(pexpr)), ylab = quote({K^{-1}}(u)),
        main = "qK(p, method = *)")
legend("topleft", col = cols, lwd = 2*7:1, lty = 1:7, bty = "n", inset = .03,
       legend= paste0("method= ",
             sQuote(c("default", "simple", "sort",
                  "discrete(1e4)", "monoH.FC(500)", "monoH.FC(5e3)", "monoH.FC(*^2)"))))

## See they *are* inverses  (but only approximately!):
eqInv <- function(qK) all.equal(p, pK(qK, cop@copula, d=d), tol=0)

eqInv(qK0 ) # "default"        0.03  worst
eqInv(qK1 ) # "simple"        0.0011 - best
eqInv(qK1.) # "simple", e-12   0.00000 (8.73 e-13) !
eqInv(qK2 ) # "sort"        0.0013 (close)
```

```
eqInv(qK2.) # "sort", e-12     0.00000 (7.32 e-12)
eqInv(qK3 ) # "discrete"       0.0026
eqInv(qK4 ) # "monoH.FC(500)"  0.0095
eqInv(qK4.) # "m.H.FC(5c)e-12" 0.00963
eqInv(qK5 ) # "monoH.FC(5e3)"  0.001148
eqInv(qK5.) # "m.H.FC(5k)e-12" 0.000989
eqInv(qK6 ) # "monoH.FC(*^2)"  0.001111
eqInv(qK6.) # "m.H.FC(*^2)e-12"0.00000 (1.190 e-09)

## and ensure the differences are not too large
stopifnot(
 all.equal(qK0, qK1, tol = 1e-2) # !
 ,
 all.equal(qK1, qK2, tol = 1e-4)
 ,
 all.equal(qK2, qK3, tol = 1e-3)
 ,
 all.equal(qK3, qK4, tol = 1e-3)
 ,
 all.equal(qK4, qK0, tol = 1e-2) # !
)


stopifnot(all.equal(p, pK(qK0, cop@copula, d=d), tol = 0.04))
```

`khoudrajiCopula` *Construction of copulas using Khoudraji's device*

Description

Creates an object representing a copula constructed using *Khoudraji's device* (Khoudraji, 1995). The resulting R object is either of class "`khoudrajiBivCopula`", "`khoudrajiExplicitCopula`" or "`khoudrajiCopula`".

In the bivariate case, given two copulas C_1 and C_2, Khoudraji's device consists of defining a copula whose c.d.f. is given by:

$$C_1(u_1^{1-a_1}, u_2^{1-a_2})C_2(u_1^{a_1}, u_2^{a_2})$$

where a_1 and a_2 are *shape parameters* in [0,1].

The construction principle (see also Genest et al. 1998) is a special case of that considered in Liebscher (2008).

Usage

```
khoudrajiCopula(copula1 = indepCopula(), copula2 = indepCopula(dim = d),
                shapes = rep(NA_real_, dim(copula1)))
```

Arguments

copula1, copula2
: each a copula (possibly generalized, e.g., also a "rotCopula") of the same dimension d. By default independence copulas, where copula2 gets the dimension from copula1.

shapes
: numeric vector of length d, with values in $[0, 1]$.

Details

If the argument copulas are bivariate, an object of class "khoudrajiBivCopula" will be constructed. If they are exchangeable and d-dimensional with $d > 2$, and if they have explicit p.d.f. and c.d.f. expressions, an object of class "khoudrajiExplicitCopula" will be constructed. For the latter two classes, density evaluation is implemented, and fitting and goodness-of-fit testing can be attempted. If $d > 2$ but one of the argument copulas does not have explicit p.d.f. and c.d.f. expressions, or is not exchangeable, an object of class "khoudrajiCopula" will be constructed, for which density evaluation is not possible.

Value

A new object of class "khoudrajiBivCopula" in dimension two or of class "khoudrajiExplicitCopula" or "khoudrajiCopula" when $d > 2$.

References

Genest, C., Ghoudi, K., and Rivest, L.-P. (1998), Discussion of "Understanding relationships using copulas", by Frees, E., and Valdez, E., *North American Actuarial Journal* **3**, 143–149.

Khoudraji, A. (1995), Contributions à l'étude des copules et àla modélisation des valeurs extrêmes bivariées, *PhD thesis, Université Laval*, Québec, Canada.

Liebscher, E. (2008), Construction of asymmetric multivariate copulas, *Journal of Multivariate Analysis* **99**, 2234–2250.

Examples

```
## A bivariate Khoudraji-Clayton copula
kc <- khoudrajiCopula(copula2 = claytonCopula(6),
                      shapes = c(0.4, 0.95))
class(kc) # "kh..._Biv_Copula"
kc
contour(kc, dCopula, nlevels = 20, main = "dCopula(<khoudrajiBivCopula>)")

## A Khoudraji-Clayton copula with second shape parameter fixed
kcf <- khoudrajiCopula(copula2 = claytonCopula(6),
                       shapes = fixParam(c(0.4, 0.95), c(FALSE, TRUE)))
kcf. <- setTheta(kcf, c(3, 0.2)) # (change *free* param's only)
validObject(kcf) & validObject(kcf.)

## A "nested" Khoudraji bivariate copula
kgkcf <- khoudrajiCopula(copula1 = gumbelCopula(3),
                         copula2 = kcf,
```

```
                         shapes = c(0.7, 0.25))
kgkcf # -> 6 parameters (1 of 6 is 'fixed')
contour(kgkcf, dCopula, nlevels = 20,
        main = "dCopula(<khoudrajiBivC.(nested)>)")

(Xtras <- copula:::doExtras()) # determine whether examples will be extra (long)
n <- if(Xtras) 300 else 64 # sample size (realistic vs short for example)

u <- rCopula(n, kc)
plot(u)

## For likelihood (or fitting), specify the "free" (non-fixed) param's:
##           C1:  C2c C2s1    sh1  sh2
loglikCopula(c(3,   6, 0.4,   0.7, 0.25),
             u = u, copula = kgkcf)

## Fitting takes time (using numerical differentiation) and may be difficult:

## Starting values are required for all parameters
f.IC <- fitCopula(khoudrajiCopula(copula2 = claytonCopula()),
                  start = c(1.1, 0.5, 0.5), data = pobs(u),
                  optim.method = "Nelder-Mead")
summary(f.IC)
confint(f.IC) # (only interesting for reasonable sample size)

## Because of time,  don't run these by default :

## Second shape parameter fixed to 0.95
kcf2 <- khoudrajiCopula(copula2 = claytonCopula(),
                        shapes = fixParam(c(NA_real_, 0.95), c(FALSE, TRUE)))
system.time(
f.ICf <- fitCopula(kcf2, start = c(1.1, 0.5), data = pobs(u),
                   optim.method = "Nelder-Mead")
) # ~ 7-8 sec
confint(f.ICf) # !
coef(f.ICf, SE=TRUE)

## With a different optimization method
system.time(
f.IC2 <- fitCopula(kcf2, start = c(1.1, 0.5), data = pobs(u),
                   optim.method = "BFGS")
)
printCoefmat(coef(f.IC2, SE=TRUE), digits = 3) # w/o unuseful extra digits


if(Xtras >= 2) { # really S..L..O..W... --------

## GOF example
optim.method <- "Nelder-Mead" #try "BFGS" as well
gofCopula(kcf2, x = u, start = c(1.1, 0.5), optim.method = optim.method)
gofCopula(kcf2, x = u, start = c(1.1, 0.5), optim.method = optim.method,
          sim = "mult")
## The goodness-of-fit tests should hold their level
```

```
## but this would need to be tested

## Another example under the alternative
u <- rCopula(n, gumbelCopula(4))
gofCopula(kcf2, x = u, start = c(1.1, 0.5), optim.method = optim.method)
gofCopula(kcf2, x = u, start = c(1.1, 0.5), optim.method = optim.method,
          sim = "mult")

}## ------ end { really slow gofC*() } --------

## Higher-dimensional constructions

## A three dimensional Khoudraji-Clayton copula
kcd3 <- khoudrajiCopula(copula1 = indepCopula(dim=3),
                        copula2 = claytonCopula(6, dim=3),
                        shapes = c(0.4, 0.95, 0.95))

n <- if(Xtras) 1000 else 100 # sample size (realistic vs short for example)
u <- rCopula(n, kcd3)
splom2(u)
v <- matrix(runif(15), 5, 3)
dCopula(v, kcd3)

## A four dimensional Khoudraji-Normal copula
knd4 <- khoudrajiCopula(copula1 = indepCopula(dim=4),
                        copula2 = normalCopula(.9, dim=4),
                        shapes = c(0.4, 0.95, 0.95, 0.95))
knd4
stopifnot(class(knd4) == "khoudrajiCopula")
u <- rCopula(n, knd4)
splom2(u)
## TODO :
## dCopula(v, knd4) ## not implemented
```

---

khoudrajiCopula-class *Class* "khoudrajiCopula" *and its Subclasses*

Description

The *virtual* class "asymCopula" of (conceptually) all asymmetric copulas and its 'subclass' "asym2copula" of those which are constructed from two copulas.

More specifically, the class "khoudrajiCopula" and its two subclasses "khoudrajiBivCopula" and "khoudrajiExplicitCopula" represent copulas constructed using Khoudraji's device from two copulas of the same dimension; see khoudrajiCopula() for more details.

Objects from the Class

Objects are typically created via khoudrajiCopula(...).

Slots

As these classes extend "copula", they have all its slots: dimension, parameters, param.names, param.lowbnd, param.upbnd, and fullname. The classes "khoudrajiCopula" and "khoudrajiBivCopula" have the extra slots

copula1: object of class "copula".

copula2: second object of class "copula".

In addition to these, the class "khoudrajiExplicitCopula" has the slots

exprdist: an expression, ...

derExprs1: an expression of length d, ...

derExprs2: an expression of length d, ...

Methods

When possible, methods are defined at the "khoudrajiCopula" class level. The implementation of method dCopula for instance is however not possible at that level. In addition, it differs for "khoudrajiBivCopula" and "khoudrajiExplicitCopula" classes.

References

Genest, C., Ghoudi, K., and Rivest, L.-P. (1998), Discussion of "Understanding relationships using copulas", by Frees, E., and Valdez, E., *North American Actuarial Journal* **3**, 143–149.

Khoudraji, A. (1995), Contributions à l'étude des copules et àla modélisation des valeurs extrêmes bivariées, *PhD thesis, Université Laval*, Québec, Canada.

Liebscher, E. (2008), Construction of asymmetric multivariate copulas, *Journal of Multivariate Analysis* **99**, 2234–2250.

See Also

khoudrajiCopula()

Examples

```
showClass("khoudrajiCopula")# two subclasses

## all methods applicable to these subclasses:
(meths <- sapply(names(getClass("khoudrajiCopula")@subclasses),
                 function(CL) methods(class = CL),
                 simplify=FALSE))
```

`log1mexp` *Compute f(a) =* log*(1 +/-* exp*(-a)) Numerically Optimally*

Description

Compute f(a) = log(1 - exp(-a)), respectively g(x) = log(1 + exp(x)) quickly numerically accurately.

Usage

```
log1mexp(a, cutoff = log(2))
log1pexp(x, c0 = -37, c1 = 18, c2 = 33.3)
```

Arguments

`a`	numeric vector of positive values
`x`	numeric vector
`cutoff`	positive number; `log(2)` is "optimal", but the exact value is unimportant, and anything in $[0.5, 1]$ is fine.
`c0, c1, c2`	cutoffs for `log1pexp`; see below.

Value

f(a) == log(1 - exp(-a)) == log1p(-exp(-a)) == log(-expm1(-a))

or

g(x) == log(1 + exp(x)) == log1p(exp(x))

computed accurately and quickly

References

Martin Mächler (2012). Accurately Computing $\log(1-\exp(-|a|))$; https://CRAN.R-project.org/package=Rmpfr/vignettes/log1mexp-note.pdf.

Examples

```
a <- 2^seq(-58,10, length = 256)
fExpr <- expression(
          log(1 - exp(-a)),
          log(-expm1(-a)),
          log1p(-exp(-a)),
          log1mexp(a))
names(fExpr) <- c("DEF", "expm1", "log1p", "F")
str(fa <- do.call(cbind, as.list(fExpr)))
head(fa)# expm1() works here
tail(fa)# log1p() works here

## graphically:
lwd <- 1.5*(5:2); col <- adjustcolor(1:4, 0.4)
```

```
op <- par(mfcol=c(1,2), mgp = c(1.25, .6, 0), mar = .1+c(3,2,1,1))
  matplot(a, fa, type = "l", log = "x", col=col, lwd=lwd)
  legend("topleft", fExpr, col=col, lwd=lwd, lty=1:4, bty="n")
  # expm1() & log1mexp() work here

  matplot(a, -fa, type = "l", log = "xy", col=col, lwd=lwd)
  legend("left", paste("-",fExpr), col=col, lwd=lwd, lty=1:4, bty="n")
  # log1p() & log1mexp() work here
par(op)

curve(log1pexp, -10, 10, asp=1)
abline(0,1, h=0,v=0, lty=3, col="gray")

## Cutoff c1 for log1pexp() -- not often "needed":
curve(log1p(exp(x)) - log1pexp(x), 16, 20, n=2049)
## need for *some* cutoff:
x <- seq(700, 720, by=2)
cbind(x, log1p(exp(x)), log1pexp(x))

## Cutoff c2 for log1pexp():
curve((x+exp(-x)) - x, 20, 40, n=1025)
curve((x+exp(-x)) - x, 33.1, 33.5, n=1025)
```

`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 of the following 4 variables:

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

`alae` a numeric vector of the corresponding 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)
```

math-fun	*Sinc, Zolotarev's, and Other Mathematical Utility Functions*

Description

`sinc(x)` computes the *sinc function* $s(x) = \sin(x)/x$ for $x \neq 0$ and $s(0) = 1$, such that $s()$ is continuous, also at $x = 0$.

`A..Z(x, a)` computes Zolotarev's function to the power `1-a`.

Usage

```
sinc(x)
A..Z(x, alpha, I.alpha = 1 - alpha)
```

Arguments

`x`	`numeric` argument in $[0, \pi]$, typically a vector.
`alpha`	parameter in (0,1].
`I.alpha`	must be `= 1 - alpha`, maybe more accurately when `alpha` is very close to 1.

Details

For more details about Zolotarev's function, see, for example, Devroye (2009).

Value

`A..Z(x,alpha)` is $\tilde{A}_Z(x, \alpha)$, defined as

$$\frac{\sin(\alpha x)^\alpha \sin((1-\alpha)x)^{1-\alpha}}{\sin(x)}, \; x \in [0, \pi],$$

where $\alpha \in (0, 1]$ is `alpha`.

References

Devroye, L. (2009) Random variate generation for exponentially and polynomially tilted stable distributions, *ACM Transactions on Modeling and Computer Simulation* **19**, 18, 1–20.

See Also

`retstable` internally makes use of these functions.

Examples

```
curve(sinc, -15,25); abline(h=0,v=0, lty=2)
curve(A..Z(x, 0.25), xlim = c(-4,4),
      main = "Zolotarev's function A(x) ^ 1-alpha")
```

matrix_tools *Tools to Work with Matrices*

Description

p2P() creates a matrix from a given vector of parameters. P2p() creates a numeric vector from a given matrix, currently useful for elliptical copulas.

getSigma() returns the $d \times d$ symmetric matrix Σ which is called "Rho" as well, written (capital Greek ρ !) as P (and hence sometimes erronously pronounced "Pee"). Note that getSigma() works for all elliptical copulas and uses p2P() for the "unstuctured" case, dispstr = "un".

extremePairs() identifies pairs with the largest (or smallest or both) entries in a symmetric matrix.

Usage

```
p2P(param, d = floor(1 + sqrt(2*length(param))))
P2p(P)
getSigma(copula)
extremePairs(x, n = 6, method = c("largest", "smallest", "both"),
             use.names = FALSE)
```

Arguments

param	a parameter vector.
d	dimension of the resulting matrix. The default is correct under the assumption (of p2P() in general!) that param is the lower-triangular part of a correlation matrix P and hence corresponds to ellipCopula(.., dispstr = "un").
P	a matrix which should be converted to a vector.
copula	an **elliptical** copula, i.e., an object (extending) class ellipCopula; typically resulting from tCopula() or normalCopula().
x	a symmetric matrix.
n	the number of pairs with smallest (or largest) values to be displayed.
method	a character string indicating the method to be used (with "largest" to comute the n pairs with largest entries in x (sorted in decreasing order); with "smallest" to compute the n pairs with smallest entries in x (sorted in increasing order); and with "both" to comute the 2n pairs with n largest entries and n smallest entries (sorted in decreasing order)).
use.names	A logical indicating whether colnames(x) are used as labels (if !is.null(colnames(x))).

Details

These auxiliary functions are often used when working with elliptical copulas.

Value

p2P: a symmetric matrix with ones on the diagonal and the values of `param` filled column-wise below the diagonal (which corresponds to row-wise filling above the diagonal).

P2p: vector of column-wise below-diagonal entries of `P` (equal to the row-wise above-diagonal entries in case of a symmetric matrix).

getSigma: matrix as from `p2P()` for all cases of elliptical copulas.

extremePairs: a data.frame consisting of three columns (`row` (index or name), `col` (index or name), `value`).

See Also

ellipCopula, tCopula, normalCopula.

Examples

```
## display the definitions
p2P
P2p
extremePairs

param <- (2:7)/10
tC <- tCopula(param, dim = 4, dispstr = "un", df = 3)
## consistency of the three functions :
P <- p2P(param) # (using the default 'd')
stopifnot(identical(param, P2p(P)),
  identical(P, getSigma(tC)))

## Toeplitz case:
(tCt <- tCopula((2:6)/10, dim = 6, disp = "toep"))
(rhoP <- tCt@getRho(tCt))
stopifnot(identical(getSigma (tCt),
    toeplitz (c(1, rhoP))))

## "AR1" case:
nC.7 <- normalCopula(0.8, dim = 7, dispstr = "ar1")
(Sar1.7 <- getSigma(nC.7))
0.8^(0:(7-1)) #  1  0.8  0.64  0.512 ..
stopifnot(all.equal(Sar1.7, toeplitz(0.8^(0:(7-1)))))
```

---

mixCopula — *Create Mixture of Copulas*

Description

A mixture of m copulas of dimension d with weights w_j, $j = 1, 2, \dots, m$ is itself a d-dimensional copula, with cumulative distribution function

$$C(x) = \sum_{j=1}^{m} w_j C_j(x),$$

and (probability) density function

$$c(x) = \sum_{j=1}^{m} w_j c_j(x),$$

where C_j are the CDFs and c_j are the densities of the m component copulas, $j = 1, 2, \ldots, m$.

Usage

```
mixCopula(coplist, w = NULL)
```

Arguments

coplist	a list of length $m (\geq 1)$ copulas (each inheriting from parCopula), all of the same dimension.
w	numeric vector of length m of non-negative mixture weights, or NULL, which means equal weights.

Details

It easy to see that the tail dependencies lambda() and Spearman's rank correlation rho() can be computed as mixture of the individual measures.

Value

an object of class mixCopula

See Also

khoudrajiCopula, rotCopula also create new copula models from existing ones.

Examples

```
mC <- mixCopula(list(gumbelCopula(2.5, dim=3),
                     claytonCopula(pi, dim=3),
                     tCopula(0.7, dim=3)),
                c(2,2,4)/8)
mC
stopifnot(dim(mC) == 3)

set.seed(17)
uM <- rCopula(600, mC)
splom2(uM, main = "mixCopula( (gumbel, clayton, t-Cop) )")
d.uM <- dCopula(uM, mC)
p.uM <- pCopula(uM, mC)

## mix a Gumbel with a rotated Gumbel (with equal weights 1/2):
mGG <- mixCopula(list(gumbelCopula(2), rotCopula(gumbelCopula(1.5))))
rho(mGG)   # 0.57886
lambda(mGG)# both lower and upper tail dependency
```

```
loglikCopula(c(2.5, pi, rho.1=0.7, df = 4, w = c(2,2,4)/8),
                    u = uM, copula = mC)
ll.df <- Vectorize(function(df, rho)
                   loglikCopula(c(2.5, pi, rho.1=rho, df=df, w = c(2,2,4)/8),
                                uM, mC))
(df. <- 1/rev(seq(1/8, 1/2, length=21)))# in [2, 8] equidistant in 1/. scale

ll. <- ll.df(df., rho = (rh1 <- 0.7))
plot(df., ll., type = "b", main = "loglikCopula((.,.,rho = 0.7, df, ..), u, <mixCopula>)")

if(!exists("Xtras")) Xtras <- copula:::doExtras() ; cat("Xtras: ", Xtras,"\n")
if (Xtras) {
  Rhos <- seq(0.55, 0.70, by = 0.01)
  ll.m <- matrix(NA, nrow=length(df.), ncol=length(Rhos))
  for(k in seq_along(Rhos)) ll.m[,k] <- ll.df(df., rho = Rhos[k])
  tit <- "loglikelihood(<tCop>, true param. for rest)"
  persp         (df., Rhos, ll.m, phi=30, theta = 50, ticktype="detailed", main = tit)
  filled.contour(df., Rhos, ll.m, xlab="df", ylab = "rho", main = tit)
}

## fitCopula() example with "fixed weights" -- (only these are "ok" !!) -------------

(mNt <- mixCopula(list(normalCopula(0.95), tCopula(-0.7)), w = c(1, 2) / 3))
set.seed(1452) ; U <- pobs(rCopula(1000, mNt))
(m1 <- mixCopula(list(normalCopula(), tCopula()), w = mNt@w))

getTheta(m1, freeOnly = TRUE, attr = TRUE)
getTheta(m1, named=TRUE)
copula:::isFree(m1)
fixedParam(m1) <- fx <- c(FALSE, FALSE, FALSE, TRUE, TRUE)
stopifnot(identical(copula:::isFree(m1), !fx))

if (Xtras) { ## time
  print(system.time(
    fit <- fitCopula(m1, start = c(0, 0, 10), data = U)))
  ## 16 sec (nb-mm4)
  print( fit )
  print( summary(fit) )#-> incl  'Std.Error' (which seems small for rho1 !)
} #{Xtras}
```

---

mixCopula-class *Class* "mixCopula" *of Copula Mixtures*

Description

The class "mixCopula" is the class of all finite mixtures of copulas.

These are given by (a list of) m "arbitrary" copulas, and their respective m non-negative probability weights.

Objects from the Class

Objects are typically created by `mixCopula()`.

Slots

`w`**:** Object of class `"mixWeights"`, basically a non-negative `numeric` vector of length, say m, which sums to one.

`cops`**:** Object of class `"parClist"`, a `list` of (parametrized) copulas, `"parCopula"`.

Extends

Class `"parCopula"`, directly. Class `"Copula"`, by class "parCopula", distance 2.

Methods

dim `signature(x = "mixCopula")`: dimension of copula.

rho `signature(x = "mixCopula")`: Spearman's rho of copula `x`.

lambda `signature(x = "mixCopula")`: lower and upper tail dependecies `lambda`, $(\lambda[L], \lambda[U])$, of the mixture copula.

Note

As the probability weights must some to one (`1`), which is part of the validity (see `validObject`) of an object of class `"mixWeights"`, the number of "free" parameters inherently is (at most) one *less* than the number of mixture components m.

Because of that, it does not make sense to fix (see `fixParam` or `fixedParam<-`) all but one of the weights: Either all are fixed, or at least two must be free. Further note, that the definition of free or fixed parameters, and the meaning of the methods (for `mixCopula`) of `getTheta`, `setTheta` and `fixedParam<-` will probably change in a next release of package **copula**, where it is planned to use a reparametrization better suited for `fitCopula`.

See Also

`mixCopula` for creation and examples.

Examples

```
showClass("mixCopula")
```

multIndepTest *Independence Test Among Continuous Random Vectors Based on the Empirical Copula Process*

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,
              verbose = interactive())
```

Arguments

`x`	data frame (`data.frame`) or 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 `sum(d) == ncol(x)`.
`m`	maximum cardinality of the subsets of random vectors for which a test statistic is to be computed. It makes sense to consider `m << 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.
`verbose`	a logical specifying if progress should be displayed via `txtProgressBar`.

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).

The former argument `print.every` is deprecated and not supported anymore; use `verbose` instead.

References

Deheuvels, P. (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.

Deheuvels, P. (1981), A non parametric test for independence, *Publ. Inst. Statist. Univ. Paris*. **26**, 29–50.

Genest, C. and Rémillard, B. (2004), Tests of independence and randomness based on the empirical copula process. *Test* **13**, 335–369.

Genest, C., Quessy, J.-F., and Rémillard, B. (2006). Local efficiency of a Cramer-von Mises test of independence, *Journal of Multivariate Analysis* **97**, 274–294.

Genest, C., Quessy, J.-F., and Rémillard, B. (2007), Asymptotic local efficiency of Cramér-von Mises tests for multivariate independence. *The Annals of Statistics* **35**, 166–191.

Kojadinovic, I. and Holmes, M. (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(n, nc),rCopula(n, nc))

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 Time Series via Empirical Copula*

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,
                    verbose = interactive())
```

Arguments

x	data frame or matrix of 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 << 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.
verbose	a logical specifying if progress should be displayed via `txtProgressBar`.

Details

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

The former argument `print.every` is deprecated and not supported anymore; use `verbose` instead.

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

Deheuvels, P. (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.

Deheuvels, P. (1981) A non parametric test for independence. *Publ. Inst. Statist. Univ. Paris* **26**, 29–50.

Genest, C. and Rémillard, B. (2004) Tests of independence and randomness based on the empirical copula process. *Test* **13**, 335–369.

Ghoudi, K., Kulperger, R., and Rémillard, B. (2001) A nonparametric test of serial independence for times series and residuals. *Journal of Multivariate Analysis* **79**, 191–218.

Kojadinovic, I. and Yan, J. (2011) Tests of multivariate serial independence based on a Möbius decomposition of the independence empirical copula process. *Annals of the Institute of Statistical Mathematics* **63**, 347–373.

See Also

serialIndepTest, indepTest, multIndepTest, dependogram

Examples

```
## A multivariate time series {minimal example for demo purposes}
d <- 2
n <- 100 # sample size *and* "burn-in" size
param <- 0.25
A <- matrix(param,d,d) # the bivariate AR(1)-matrix
set.seed(17)
ar <- matrix(rnorm(2*n * d), 2*n,d) # used as innovations
for (i in 2:(2*n))
  ar[i,] <- A %*% ar[i-1,] + ar[i,]
## drop burn-in :
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 and *parametric* margins.

For likelihood and fitting these distributions to data, see fitMvdc.

Usage

```
mvdc(copula, margins, paramMargins, marginsIdentical = FALSE,
     check = TRUE, fixupNames = TRUE)
dMvdc(x, mvdc, log=FALSE)
pMvdc(x, mvdc)
rMvdc(n, mvdc)
```

Arguments

`copula`	an object of "`copula`".
`margins`	a `character` vector specifying all the parametric marginal distributions. See details below.
`paramMargins`	a `list` whose each component is a list (or numeric vectors) of named components, giving the parameter values of the marginal distributions. See details below.
`marginsIdentical`	logical variable restricting the marginal distributions to be identical.
`check`	logical indicating to apply quick checks about existence of `margins` "p*" and "d*" functions.
`fixupNames`	logical indicating if the parameters of the margins should get automatic names (from `formals`(p<mar_i>)).
`mvdc`	a "`mvdc`" object.
`x`	a numeric vector of length the copula dimension, say d, or a matrix with the number of columns being d, giving the coordinates of the points where the density or distribution function needs to be evaluated.
`log`	logical indicating if the `log` density should be returned.
`n`	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, the list

```
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 cumulative distribution function, and `rMvdc()` generates random variates.

Note

mvdc(), fitMvdc, etc, are only for *parametric* margins. If you do not want to model all margins parametrically, use the standard copula approach, transforming the data by their empirical margins via pobs and modelling the copula alone, e.g., using fitCopula, i.e., conceptually, using

```
fitCopula(.., pobs(x))
```

See Also

ellipCopula, archmCopula; the classes mvdc and copula.

Examples

```
## construct a bivariate distribution whose marginals
## are normal and exponential respectively, coupled
## together via a normal copula
mv.NE <- mvdc(normalCopula(0.75), c("norm", "exp"),
              list(list(mean = 0, sd =2), list(rate = 2)))
dim(mv.NE)
mv.NE  # using its print() / show() method

persp  (mv.NE, dMvdc, xlim = c(-4, 4), ylim=c(0, 2), main = "dMvdc(mv.NE)")
persp  (mv.NE, pMvdc, xlim = c(-4, 4), ylim=c(0, 2), main = "pMvdc(mv.NE)")
contour(mv.NE, dMvdc, xlim = c(-4, 4), ylim=c(0, 2))

# Generate (bivariate) random numbers from that, and visualize
x.samp <- rMvdc(250, mv.NE)
plot(x.samp)
summary(fx <- dMvdc(x.samp, mv.NE))
summary(Fx <- pMvdc(x.samp, mv.NE))
op <- par(mfcol=c(1,2))
pp <- persp(mv.NE, pMvdc, xlim = c(-5,5), ylim=c(0,2),
            main = "pMvdc(mv.NE)", ticktype="detail")

px <- copula:::perspMvdc(x.samp, FUN = F.n, xlim = c(-5, 5), ylim = c(0, 2),
                         main = "F.n(x.samp)", ticktype="detail")
par(op)
all.equal(px, pp)# about 5% difference
```

---

mvdc-class *Class "mvdc"*

Description

Class representing multivariate distributions constructed using Sklar's theorem.

Objects from the Class

Objects are typically created by mvdc(), or can be created by calls of the form new("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"): ...

dim signature(x = "mvdc"): the dimension of the distribution; this is the same as dim(x@copula).

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

show signature(object = "mvdc"): quite compactly display the content of the "mvdc" object.

See Also

mvdc, also for examples; for fitting, fitMvdc.

---

nacFrail.time *Timing for Sampling Frailties of Nested Archimedean Copulas*

Description

This function provides measurements of user run times for the frailty variables involved in a nested Archimedean copula.

Usage

```
nacFrail.time(n, family, taus, digits = 3, verbose = FALSE)
```

Arguments

n	integer specifying the sample size to be used for the random variates V_0 and V_{01}.
family	the Archimedean family (class "acopula") for which V_0 and V_{01} are sampled.
taus	numeric vector of Kendall's taus. This vector is converted to a vector of copula parameters θ, which then serve as θ_0 and θ_1 for a three-dimensional fully nested Archimedean copula of the specified family. First, for each θ_0, n random variates V_0 are generated. Then, given the particular θ_0 and the realizations V_0, n random variates V_{01} are generated for each θ_1 fulfilling the sufficient nesting condition; see paraConstr in acopula.
digits	number of digits for the output.
verbose	logical indicating if nacFrail.time output should generated while the random variates are generated (defaults to FALSE).

Value

A $k \times k$ matrix of user run time measurements in milliseconds (`1000*system.time(.)[1]`) where k is `length(taus)`. The first column contains the run times for generating the V_0s. For the submatrix that remains if the first column is removed, row i (for θ_{0_i}) contains the run times for the V_{01}s for a particular θ_0 and all the admissible θ_1s.

See Also

The class `acopula` and our predefined `"acopula"` family objects in `acopula-families`. For some timings on a standard notebook, see `demo(timings)` (or the file '`timings.R`' in the demo folder).

Examples

```
## takes about 7 seconds:% so we rather test a much smaller set in R CMD check

nacFrail.time(10000, "Gumbel", taus=  c(0.05,(1:9)/10, 0.95))

system.time(
print( nacFrail.time(1000,  "Gumbel", taus = c(0.5,1,6,9)/10) )
)
```

---

nacopula-class   *Class "nacopula" of Nested Archimedean Copulas*

Description

Class of nested Archimedean Copulas, `"nacopula"`, and its *specification* `"outer_nacopula"` differ only by the validation method, which is stricter for the outer(most) copula (the root copula).

Objects from the Class

Objects can be created by calls of the form `new("nacopula", ...)`, which is only intended for experts. Root copulas are typically constructed by `onacopula(.)`.

Slots

`copula`: an object of class `"acopula"`, denoting the top-level Archimedean copula of the nested Archimedean copula, that is, the root copula.

`comp`: an `integer` vector (possibly of length 0) of indices of components in `1:d` which are not nested Archimedean copulas. Here, d denotes the *dimension* of the random vectors under consideration; see the `dim()` method below.

`childCops`: a (possibly empty) `list` of further nested Archimedean copulas (child copulas), that is, objects of class `"nacopula"`. The `"nacopula"` objects therefore contain `"acopula"` objects as special cases.

Methods

dim `signature(x = "nacopula")`: returns the dimension d of the random vector U following `x`.

show `signature("nacopula")`: calling `printNacopula` for a compact overview of the nested Archimedean copula under consideration.

See Also

`onacopula` for building (outer) "nacopula" objects. For the class definition of the `copula` component, see `acopula`.

Examples

```
## nacopula and outer_nacopula class information
showClass("nacopula")
showClass("outer_nacopula")

## Construct a three-dimensional nested Frank copula with parameters
## chosen such that the Kendall's tau of the respective bivariate margins
## are 0.2 and 0.5.
theta0 <- copFrank@iTau(.2)
theta1 <- copFrank@iTau(.5)
C3 <- onacopula("F", C(theta0, 1, C(theta1, c(2,3))))

C3 # displaying it, using show(C3); see help(printNacopula)

## What is the dimension of this copula?
dim(C3)

## What are the indices of direct components of the root copula?
C3@comp

## How does the list of child nested Archimedean copulas look like?
C3@childCops # only one child for this copula, components 2, 3
```

`nacPairthetas` *Pairwise Thetas of Nested Archimedean Copulas*

Description

Return a $d * d$ matrix of pairwise thetas for a nested Archimedean copula (`nacopula`) of dimension d.

Usage

```
nacPairthetas(x)
```

Arguments

x an (outer) nacopula (with thetas sets).

Value

a $(d \times d)$ matrix of thetas, say T, where T[j,k] = theta of the bivariate Archimedean copula $C(U_j, U_k)$.

See Also

the class nacopula (with its dim method).

Examples

```
## test with
options(width=97)

(mm <- rnacModel("Gumbel", d=15, pr.comp = 0.25, order="random"))
stopifnot(isSymmetric(PT <- nacPairthetas(mm)))
round(PT, 2)

## The tau's -- "Kendall's correlation matrix" :
round(copGumbel@tau(PT), 2)

## do this several times:
m1 <- rnacModel("Gumbel", d=15, pr.comp = 1/8, order="seq")
stopifnot(isSymmetric(PT <- nacPairthetas(m1)))
m1; PT



m100 <- rnacModel("Gumbel", d= 100, pr.comp = 1/16, order="seq")
system.time(PT <- nacPairthetas(m100))# how slow {non-optimal algorithm}?
##-- very fast, still!
stopifnot(isSymmetric(PT))
m100

## image(PT)# not ok -- want one color per theta
nt <- length(th0 <- unique(sort(PT[!is.na(PT)])))
th1 <- c(th0[1]/2, th0, 1.25*th0[nt])
ths <- (th1[-1]+th1[-(nt+2)])/2
image(log(PT), breaks = ths, col = heat.colors(nt))

## Nicer and easier:
require(Matrix)
image(as(log(PT),"Matrix"), main = "log( nacPairthetas( m100 ))",
      useAbs=FALSE, useRaster=TRUE, border=NA)
```

`nesdepth` *Nesting Depth of a Nested Archimedean Copula ("nacopula")*

Description

Compute the nesting depth of a nested Archimedean copula which is the length of the longest branch in the tree representation of the copula, and hence at least one.

Usage

```
nesdepth(x)
```

Arguments

`x`	object of class "`nacopula`".

Value

an integer, the nesting depth of the nested Archimedean copula. An (unnested) Archimedean copula has depth `1`.

See Also

`dim` of nacopulas.

Examples

```
F2 <- onacopula("F", C(1.9, 1, C(4.5, c(2,3))))
F2
F3 <- onacopula("Clayton", C(1.5, 3:1,
                             C(2.5, 4:5,
                               C(15, 9:6))))
nesdepth(F2) # 2
nesdepth(F3) # 3
```

`onacopula` *Constructing (Outer) Nested Archimedean Copulas*

Description

Constructing (outer) nested Archimedean copulas (class `outer_nacopula`) is most conveniently done via `onacopula()`, using a nested $C(...)$ notation.

Slightly less conveniently, but with the option to pass a `list` structure, `onacopulaL()` can be used, typically from inside another function programmatically.

Usage

```
onacopula (family, nacStructure)
onacopulaL(family, nacList)
nac2list(x)
```

Arguments

`family` either a `character` string, the short or longer form of the Archimedean family name (for example, "Clayton" or simply "C"); see the `acopula-families` documentation, or an `acopula` family object.

`nacStructure` a "formula" of the form

$$C(\theta, c(i_1, \ldots, i_c), \mathrm{list}(C(..), ..., C(..))).$$

Note that $C()$ has (maximally) three arguments: the first is the copula parameter (vector) θ, the second a (possibly empty) vector of integer indices of components (for the `comp` slot in `nacopula`s), and finally a (possibly empty) list of child copulas, each specified with in the $C(..)$ notation themselves.

`nacList` a `list` of length 3 (or 2), with elements

1. `theta`: θ
2. `comp`: components $c(i_1, \ldots, i_c)$
3. `children`: a list which must be a `nacList` itself and may be missing to denote the empty `list()`.

`x` an "`nacopula`", (typically "`outer_nacopula`") object.

Value

`onacopula[L]()`: An outer nested Archimedean copula object, that is, of class "`outer_nacopula`".

`nac2list`: a `list` exactly like the `naclist` argument to `onacopulaL`.

References

Those of the Archimedean families, for example, `copGumbel`.

See Also

The class definitions "`nacopula`", "`outer_nacopula`", and "`acopula`".

Examples

```
## Construct a ten-dimensional Joe copula with parameter such that
## Kendall's tau equals 0.5
theta <- copJoe@iTau(0.5)
C10 <- onacopula("J",C(theta,1:10))

## Equivalent construction with onacopulaL():
C10. <- onacopulaL("J",list(theta,1:10))
```

```
stopifnot(identical(C10, C10.),
          identical(nac2list(C10), list(theta, 1:10)))

## Construct a three-dimensional nested Gumbel copula with parameters
## such that Kendall's tau of the respective bivariate margins are 0.2
## and 0.5.
theta0 <- copGumbel@iTau(.2)
theta1 <- copGumbel@iTau(.5)
C3 <- onacopula("G", C(theta0, 1, C(theta1, c(2,3))))

## Equivalent construction with onacopulaL():
str(NAlis <- list(theta0, 1, list(list(theta1, c(2,3)))))
C3. <- onacopulaL("Gumbel", NAlis)
stopifnot(identical(C3, C3.))

## An exercise: assume you got the copula specs as character string:
na3spec <- "C(theta0, 1, C(theta1, c(2,3)))"
na3call <- parse(text = na3spec)[[1]]
C3.s <- onacopula("Gumbel", na3call)
stopifnot(identical(C3, C3.s))

## Good error message if the component ("coordinate") indices are wrong
## or do not match:
err <- try(onacopula("G", C(theta0, 2, C(theta1, c(3,2)))))

## Compute the probability of falling in [0,.01]^3 for this copula
pCopula(rep(.01,3), C3)

## Compute the probability of falling in the cube [.99,1]^3
prob(C3, rep(.99, 3), rep(1, 3))

## Construct a 6-dimensional, partially nested Gumbel copula of the form
## C_0(C_1(u_1, u_2), C_2(u_3, u_4), C_3(u_5, u_6))
theta <- 2:5
copG <- onacopulaL("Gumbel", list(theta[1], NULL, list(list(theta[2], c(1,2)),
                                                       list(theta[3], c(3,4)),
                                                       list(theta[4], c(5,6)))))
set.seed(1)
U <- rCopula(5000, copG)
pairs(U, pch=".", gap=0, labels = as.expression( lapply(1:dim(copG),
                                     function(j) bquote(italic(U[.(j)]))) ))
```

---

opower *Outer Power Transformation of Archimedean Copulas*

Description

Build a new Archimedean copula by applying the outer power transformation to a given Archimedean copula.

Usage

```
opower(copbase, thetabase)
```

Arguments

copbase	a "base" copula, that is, a copula of class acopula. Must be one of the predefined families.
thetabase	the univariate parameter θ for the generator of the base copula copbase. Hence, the copula which is transformed is fixed, that is, does not depend on a parameter.

Value

a new acopula object, namely the outer power copula based on the provided copula family copbase with fixed parameter thetabase. The transform introduces a parameter theta, so one obtains a parametric Archimedean family object as return value.

The environment of all function slots contains objects cOP (which is the outer power copula itself), copbase, and thetabase.

References

Hofert, M. (2010), *Sampling Nested Archimedean Copulas with Applications to CDO Pricing*, Suedwestdeutscher Verlag fuer Hochschulschriften AG & Co. KG.

See Also

The class acopula and our predefined "acopula" family objects in acopula-families.

Examples

```
## Construct an outer power Clayton copula with parameter thetabase such
## that Kendall's tau equals 0.2
thetabase <- copClayton@iTau(0.2)
opC <- opower(copClayton, thetabase) # "acopula" obj. (unspecified theta)

## Construct a 3d nested Archimedean copula based on opC, that is, a nested
## outer power Clayton copula.  The parameters theta are chosen such that
## Kendall's tau equals 0.4 and 0.6 for the outer and inner sector,
## respectively.
theta0 <- opC@iTau(0.4)
theta1 <- opC@iTau(0.6)
opC3d <- onacopulaL(opC, list(theta0, 1, list(list(theta1, 2:3))))
## or opC3d <- onacopula(opC, C(theta0, 1, C(theta1, c(2,3))))

## Compute the corresponding lower and upper tail-dependence coefficients
rbind(theta0 = c(
      lambdaL = opC@lambdaL(theta0),
      lambdaU = opC@lambdaU(theta0) # => opC3d has upper tail dependence
      ),
      theta1 = c(
      lambdaL = opC@lambdaL(theta1),
```

```
      lambdaU = opC@lambdaU(theta1) # => opC3d has upper tail dependence
      ))

## Sample opC3d
n <- 1000
U <- rnacopula(n, opC3d)

## Plot the generated vectors of random variates of the nested outer
## power Clayton copula.
splom2(U)

## Construct such random variates "by hand"
## (1) draw V0 and V01
V0  <- opC@ V0(n, theta0)
V01 <- opC@V01(V0, theta0, theta1)
## (2) build U
U <- cbind(
opC@psi(rexp(n)/V0,  theta0),
opC@psi(rexp(n)/V01, theta1),
opC@psi(rexp(n)/V01, theta1))
```

`pairs2` *Scatter-Plot Matrix ('pairs') for Copula Distributions with Nice Defaults*

Description

A version of **graphics**' package `pairs()`, particularly useful for visualizing dependence in multivariate (copula) data.

Usage

```
pairs2(x, labels = NULL, labels.null.lab = "U", ...)
```

Arguments

`x`	a numeric `matrix` or an R object for which `as.matrix(x)` returns such a matrix.
`labels`	the variable names, typically unspecified.
`labels.null.lab`	the `character` string determining the "base name" of the variable labels in case `labels` is `NULL` and `x` does not have all column names given.
`...`	further arguments, passed to `pairs()`.

Value

`invisible()`

See Also

splom2() for a similar function based on splom().

Examples

```
## Create a 100 x 7 matrix of random variates from a t distribution
## with four degrees of freedom and plot the generated data
U <- matrix(rt(700, df = 4), ncol = 7)
pairs2(U, pch = ".")
```

---

pairsRosenblatt *Plots for Graphical GOF Test via Pairwise Rosenblatt Transforms*

Description

pairsColList() creates a list containing information about colors for a given matrix of (approximate aka "pseudo") p-values. These colors are used in pairsRosenblatt() for visualizing a graphical goodness-of-fit test based on pairwise Rosenblatt transformed data.

Usage

```
pairsRosenblatt(cu.u, pvalueMat=pviTest(pairwiseIndepTest(cu.u)),
                method = c("scatter", "QQchisq", "QQgamma",
                           "PPchisq", "PPgamma", "none"),
                g1, g2, col = "B&W.contrast",
                colList = pairsColList(pvalueMat, col=col),
                main=NULL,
                sub = gpviString(pvalueMat, name = "pp-values"),
panel = NULL, do.qqline = TRUE,
                keyOpt = list(title="pp-value", rug.at=pvalueMat), ...)

pairsColList(P, pdiv = c(1e-04, 0.001, 0.01, 0.05, 0.1, 0.5),
             signif.P = 0.05, pmin0 = 1e-05, bucketCols = NULL,
             fgColMat = NULL, bgColMat = NULL, col = "B&W.contrast",
             BWcutoff = 170,
             bg.col = c("ETHCL", "zurich", "zurich.by.fog", "baby",
                        "heat", "greenish"),
             bg.ncol.gap = floor(length(pdiv)/3),
             bg.col.bottom = NULL, bg.col.top = NULL, ...)
```

Arguments

cu.u	(n, d, d)-array of pairwise Rosenblatt-transformed observations as returned by pairwiseCcop().
pvalueMat	(d, d)-matrix of p-values (or pp-values).
method	character indicating the plot method to be used. Currently possible are:

"scatter" a simple scatter plot.
"QQchisq" a Q-Q plot after a map to the χ^2 distribution.
"QQgamma" a Q-Q plot after a map to the gamma distribution.
"PPchisq" a P-P plot after a map to the χ^2 distribution.
"PPgamma" a P-P plot after a map to the gamma distribution.
"none" no points are plotted.

Note: These methods merely just set g1 and g2 correctly; see the code for more details.

g1 function from $[0, 1]^n \to [0, 1]^n$ applied to "x" for plotting in one panel.

g2 function from $[0, 1]^{n\times 2} \to [0, 1]^n$ applied to "y" for plotting in one panel.

colList list of colors and information as returned by pairsColList().

main title.

sub sub-title with a smart default containing a global (p)p-value.

panel a panel function as for pairs, or, by default, NULL, where the panel is set as points or "points + qqline" if the method is "QQ...." and do.qqline is true.

do.qqline if method = "QQ....", specify if the plot panels should also draw a qqline().

keyOpt argument passed to .pairsCond() for options for the key.

... additional arguments passed to .pairsCond() (for pairsRosenblatt()) and to heat_hcl() (for pairsColList; used to generate the color palette), see Details.

P $d \times d$ matrix of p-values.

pdiv numeric vector of strictly increasing p-values in (0,1) that determine the "buckets" for the background colors of .pairsCond() which creates the pairs-like goodness-of-fit plot.

signif.P significance level (must be an element of pdiv).

pmin0 a numeric indicating the lower endpoint of the p-value buckets if pmin is zero. If set to 0, the lowest value of the p-value buckets will also be 0.

Note that pmin0 should be in (0, min(pdiv)) when using pairsColList() for .pairsCond().

bucketCols vector of length as pdiv containing the colors for the buckets. If not specified, either bg.col.bottom and bg.col.top are used (if provided) or bg.col (if provided).

fgColMat (d, d)-matrix with foreground colors (the default will be black if the background color is bright and white if it is dark; see also BWcutoff).

bgColMat (d, d)-matrix of background colors; do not change this unless you know what you are doing.

col foreground color (defaults to "B&W.contrast" which switches black/white according to BWcutoff), passed to .pairsCond(). If colList is not specified, this color is used to construct the points' color.

BWcutoff number in (0, 255) for switching foreground color if col="B&W.contrast".

bg.col color scheme for the background colors.

bg.ncol.gap number of colors left out as "gap" for color buckets below/above signif.P (to make significance/non-significance more visible).

bg.col.bottom
: vector of length 3 containing a HCL color specification. If bg.col.bottom is provided and bucketCols is not, bg.col.bottom is used as the color for the bucket of smallest p-values.

bg.col.top
: vector of length 3 containing a HCL color specification. If bg.col.top is provided and bucketCols is not, bg.col.top is used as the color for the bucket of largest p-values.

Details

Extra arguments of pairsRosenblatt() are passed to .pairsCond(), these notably may include key, true by default, which draws a color key for the colors used as panel background encoding (pseudo) p-values.

pairsColList() is basically an auxiliary function to specify the colors used in the graphical goodness-of-fit test as conducted by pairsRosenblatt(). The latter is described in detail in Hofert and Mächler (2013). See also demo(gof_graph).

Value

pairsRosenblatt: invisibly, the result of .pairsCond().

pairsColList: a named list with components

- fgColMat matrix of foreground colors.
- bgColMat matrix of background colors (corresponding to P).
- bucketCols vector containing the colors corresponding to pvalueBuckets as described above.
- pvalueBuckets vector containing the endpoints of the p-value buckets.

References

Hofert, M. and Mächler, M. (2013) A graphical goodness-of-fit test for dependence models in higher dimensions; *Journal of Computational and Graphical Statistics*, **23**(3), 700–716.

See Also

pairwiseCcop() for the tools behind the scenes. demo(gof_graph) for examples.

Examples

```
## 2-dim example {d = 2} ===============
##
## "t" Copula with 22. degrees of freedom; and (pairwise) tau = 0.5
nu <- 2.2 # degrees of freedom
## Define the multivariate distribution
tCop <- ellipCopula("t", param=iTau(ellipCopula("t", df=nu), tau = 0.5),
                    dim=2, df=nu)
set.seed(19)
X <- qexp(rCopula(n = 400, tCop))

## H0 (wrongly): a Normal copula, with correct tau
copH0 <- ellipCopula("normal", param=iTau(ellipCopula("normal"), tau = 0.5))
```

```
## create array of pairwise copH0-transformed data columns
cu.u <- pairwiseCcop(pobs(X), copula = copH0)

## compute pairwise matrix of p-values and corresponding colors
pwIT <- pairwiseIndepTest(cu.u, N=200) # (d,d)-matrix of test results

round(pmat <- pviTest(pwIT), 3) # pick out p-values
## .286 and .077
pairsRosenblatt(cu.u, pvalueMat= pmat)



### A shortened version of   demo(gof_graph) -------------------------------

N <- 32 ## too small, for "testing"; realistically, use a larger one:
if(FALSE)
N <- 100

## 5d Gumbel copula ##########

n <- 250 # sample size
d <- 5 # dimension
family <- "Gumbel" # copula family
tau <- 0.5
set.seed(17)
## define and sample the copula (= H0 copula), build pseudo-observations
cop <- getAcop(family)
th <- cop@iTau(tau) # correct parameter value
copH0 <- onacopulaL(family, list(th, 1:d)) # define H0 copula
U. <- pobs(rCopula(n, cop=copH0))

## create array of pairwise copH0-transformed data columns
cu.u <- pairwiseCcop(U., copula = copH0)

## compute pairwise matrix of p-values and corresponding colors
pwIT <- pairwiseIndepTest(cu.u, N=N, verbose=interactive()) # (d,d)-matrix of test results
round(pmat <- pviTest(pwIT), 3) # pick out p-values
## Here (with seed=1):  no significant ones, smallest = 0.0603

## Plots ---------------------

## plain (too large plot symbols here)
pairsRosenblatt(cu.u, pvalueMat=pmat, pch=".")

## with title, no subtitle
pwRoto <- "Pairwise Rosenblatt transformed observations"
pairsRosenblatt(cu.u, pvalueMat=pmat, pch=".", main=pwRoto, sub=NULL)

## two-line title including expressions, and centered
title <- list(paste(pwRoto, "to test"),
              substitute(italic(H[0]:C~~bold("is Gumbel with"~~tau==tau.)),
                         list(tau.=tau)))
```

```
line.main <- c(4, 1.4)
pairsRosenblatt(cu.u, pvalueMat=pmat, pch=".",
                main=title, line.main=line.main, main.centered=TRUE)

## Q-Q plots -- can, in general, better detect outliers
pairsRosenblatt(cu.u, pvalueMat=pmat, method="QQchisq", cex=0.2)
```

---

persp-methods *Methods for Function 'persp' in Package 'copula'*

Description

Methods for function persp to draw perspective plots (of two dimensional distributions from package **copula**).

Usage

```
## S4 method for signature 'Copula'
persp(x, FUN, n.grid = 26, delta = 0,
      xlab = "u1", ylab = "u2",
      zlab = deparse(substitute(FUN))[1], zlim = NULL,
      theta = -30, phi = 30, expand = 0.618,
      ticktype = "detail", ...)
## S4 method for signature 'mvdc'
persp(x, FUN, xlim, ylim, n.grid = 26,
      xlab = "x1", ylab = "x2", zlab = deparse(substitute(FUN))[1],
      theta = -30, phi = 30, expand = 0.618,
      ticktype = "detail", ...)
```

Arguments

`x` a "Copula" or a "mvdc" object.

`FUN` the function to be plotted; typically dCopula or pCopula.

`n.grid` the number of grid points used in each dimension. This can be a vector of length two, giving the number of grid points used in x- and y-direction, respectively; the function FUN will be evaluated on the corresponding (x,y)-grid.

`delta` A small number in $[0, \frac{1}{2})$ influencing the evaluation boundaries. The x- and y-vectors will have the range `[0+delta, 1-delta]`, the default being `[0,1]`.

`xlim, ylim` The range of the x and y variables, respectively.

`xlab, ylab, zlab, zlim, theta, phi, expand, ticktype, ...`
Arguments for (the default method of) persp(), the ones enumerated here all with *different* defaults than there.

Value

invisible; a list with the following components:

x, y	The numeric vectors, as passed to persp.default.
z	The matrix of evaluated FUN values on the grid as passed to persp.default.
persp	the 4×4 transformation matrix returned by persp.default.

Methods

Perspective plots for both "copula" or "mvdc" objects, see x in the *Arguments* section.

See Also

The contour-methods for drawing contour lines of the same functions.

Examples

```
persp(claytonCopula(2),   pCopula, main = "CDF of claytonCopula(2)")
persp(  frankCopula(1.5), dCopula, main = "Density of frankCopula(1.5)")
persp(  frankCopula(1.5), dCopula, main = "c_[frank(1.5)](.)", zlim = c(0,2))

## Examples with negative tau:
(th1 <- iTau(amhCopula(), -0.1))
persp(amhCopula(th1), dCopula)
persp(amhCopula(th1), pCopula, ticktype = "simple") # no axis ticks
persp(  frankCopula(iTau(  frankCopula(), -0.1)), dCopula)
persp(claytonCopula(iTau(claytonCopula(), -0.1)), dCopula)
##
cCop.2 <- function(u, copula, ...) cCopula(u, copula, ...)[,2]
persp(    amhCopula(iTau(    amhCopula(), -0.1)), cCop.2, main="cCop(AMH...)[,2]")
persp(  frankCopula(iTau(  frankCopula(), -0.1)), cCop.2, main="cCop(frankC)[,2]")
## and  Clayton  also looks "the same" ...

## MVDC Examples ------------------------------------
mvNN <- mvdc(gumbelCopula(3), c("norm", "norm"),
          list(list(mean = 0, sd = 1), list(mean = 1)))
persp(mvNN, dMvdc, xlim=c(-2, 2), ylim=c(-1, 3), main = "Density")
persp(mvNN, pMvdc, xlim=c(-2, 2), ylim=c(-1, 3), main = "Cumulative Distr.")
```

---

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

Plackett, R. L. (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)
```

`plot-methods` *Methods for 'plot' in Package 'copula'*

Description

Methods for `plot()` to draw a scatter plot of a random sample from bivariate distributions from package **copula**.

Usage

```
## S4 method for signature 'Copula,ANY'
plot(x, n, xlim = 0:1, ylim = 0:1,
      xlab = quote(U[1]), ylab = quote(U[2]), ...)
## S4 method for signature 'mvdc,ANY'
plot(x, n, xlim = NULL, ylim = NULL,
      xlab = quote(X[1]), ylab = quote(X[2]), ...)
```

Arguments

`x`	a *bivariate* "`matrix`", "`data.frame`", "`Copula`" or a "`mvdc`" object.
`n`	when `x` is not matrix-like: The sample size of the random sample drawn from `x`.
`xlim, ylim`	the x- and y-axis limits.
`xlab, ylab`	the x- and y-axis labels.
`...`	additional arguments passed to `plot` methods, i.e., typically `plot.default`.

Value

invisible().

See Also

splom2() for a scatter-plot *matrix* based on splom().

Examples

```
## For 2-dim. 'copula' objects
n <- 1000 # sample size
nu <- 3 # degrees of freedom
tau <- 0.5 # Kendall's tau
th <- iTau(tCopula(df = nu), tau) # corresponding parameter
cop <- tCopula(th, df = nu) # define 2-d copula object
set.seed(271)
plot(cop, n = n)

## For 2-dim. 'mvdc' objects
mvNN <- mvdc(cop, c("norm", "norm"),
             list(list(mean = 0, sd = 1), list(mean = 1)))
plot(mvNN, n = n)
```

---

pnacopula *Evaluation of (Nested) Archimedean Copulas*

Description

For a (nested) Archimedean copula (object of class nacopula) x, pCopula(u, x) (or also currently still pnacopula(x, u)) evaluates the copula x at the given vector or matrix u.

Usage

```
## S4 method for signature 'matrix,nacopula'
pCopula(u, copula, ...)

## *Deprecated*:
pnacopula(x, u)
```

Arguments

copula, x	(nested) Archimedean copula of dimension d, that is, an object of class nacopula, typically from onacopula(..).
u	a numeric vector of length d or matrix with d columns.
...	unused: potential optional arguments passed from and to methods.

Details

The value of an Archimedean copula C with generator ψ at u is given by

$$C(\boldsymbol{u}) = \psi(\psi^{-1}(u_1) + \ldots + \psi^{-1}(u_d)),\ \boldsymbol{u} \in [0,1]^d.$$

The value of a nested Archimedean copula is defined similarly. Note that a d-dimensional copula is called *nested Archimedean* if it is an Archimedean copula with arguments possibly replaced by other nested Archimedean copulas.

Value

A `numeric` in $[0, 1]$ which is the copula evaluated at `u`. (Currently not parallelized.)

Note

`pCopula(u, copula)` is a *generic* function with methods for *all* our copula classes, see `pCopula`.

Examples

```
## Construct a three-dimensional nested Joe copula with parameters
## chosen such that the Kendall's tau of the respective bivariate margins
## are 0.2 and 0.5.
theta0 <- copJoe@iTau(.2)
theta1 <- copJoe@iTau(.5)
C3 <- onacopula("J", C(theta0, 1, C(theta1, c(2,3))))

## Evaluate this copula at the vector u
u <- c(.7,.8,.6)
pCopula(u, C3)

## Evaluate this copula at the matrix v
v <- matrix(runif(300), ncol=3)
pCopula(v, C3)

## Back-compatibility check
stopifnot(identical( pCopula (u, C3), suppressWarnings(
                    pnacopula(C3, u))),
          identical( pCopula (v, C3), suppressWarnings(
                    pnacopula(C3, v))))
```

---

pobs *Pseudo-Observations*

Description

Compute the pseudo-observations for the given data matrix.

Usage

```
pobs(x, na.last = "keep",
     ties.method = eval(formals(rank)$ties.method), lower.tail = TRUE)
```

Arguments

`x`	$n \times d$-matrix (or d-vector) of random variates to be converted to pseudo-observations.
`na.last`	string passed to `rank`; see there.
`ties.method`	string specifying how ranks should be computed if there are ties in any of the coordinate samples of `x`; passed to `rank`.
`lower.tail`	`logical` which, if `FALSE`, returns the pseudo-observations when applying the empirical marginal survival functions.

Details

Given n realizations $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id})^T$, $i \in \{1, \dots, n\}$ of a random vector $\boldsymbol{X}$, the pseudo-observations are defined via $u_{ij} = r_{ij}/(n+1)$ for $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, d\}$, where r_{ij} denotes the rank of x_{ij} among all x_{kj}, $k \in \{1, \dots, n\}$. When there are no ties in any of the coordinate samples of `x`, the pseudo-observations can thus also be computed by component-wise applying the marginal empirical distribution functions to the data and scaling the result by $n/(n+1)$. This asymptotically negligible scaling factor is used to force the variates to fall inside the open unit hypercube, for example, to avoid problems with density evaluation at the boundaries. Note that `pobs(, lower.tail=FALSE)` simply returns `1-pobs()`.

Value

`matrix` (or `vector`) of the same dimensions as `x` containing the pseudo-observations.

Examples

```
## Simple definition of the function:
pobs

## Draw from a multivariate normal distribution
d <- 10
set.seed(1)
P <- Matrix::nearPD(matrix(pmin(pmax(runif(d*d), 0.3), 0.99), ncol=d))$mat
diag(P) <- rep(1, d)
n <- 500
x <- MASS::mvrnorm(n, mu = rep(0, d), Sigma = P)

## Compute pseudo-observations (should roughly follow a Gauss
## copula with correlation matrix P)
u <- pobs(x)
plot(u[,5],u[,10], xlab=quote(italic(U)[1]), ylab=quote(italic(U)[2]))

## All components: pairwise plot
pairs(u, gap=0, pch=".", labels =
      as.expression( lapply(1:d, function(j) bquote(italic(U[.(j)]))) ))
```

polylog *Polylogarithm* Li_s(z) *and Debye Functions*

Description

Compute the polylogarithm function $\mathrm{Li_s(z)}$, initially defined as the power series,

$$\mathrm{Li}_s(z) = \sum_{k=1}^{\infty} \frac{z^k}{k^s},$$

for $|z| < 1$, and then more generally (by analytic continuation) as

$$\mathrm{Li}_1(z) = -\log(1-z),$$

and

$$\mathrm{Li}_{s+1}(z) = \int_0^z \frac{\mathrm{Li}_s(t)}{t}\,dt.$$

Currently, mainly the case of negative integer s is well supported, as that is used for some of the Archimedean copula densities.

For $s = 2$, $\mathrm{Li_2(z)}$ is also called 'dilogarithm' or "Spence's function". The `"default"` method uses the `dilog` or `complex_dilog` function from package **gsl**, respectively when $s = 2$.

Also compute the Debye_n functions, for $n = 1$ and $n = 2$, in a slightly more general manner than the **gsl** package functions `debye_1` and `debye_2` (which cannot deal with non-finite `x`.)

Usage

```
polylog(z, s,
        method = c("default", "sum", "negI-s-Stirling",
                   "negI-s-Eulerian", "negI-s-asymp-w"),
        logarithm = FALSE, is.log.z = FALSE, is.logmlog = FALSE,
        asymp.w.order = 0, n.sum)

debye1(x)
debye2(x)
```

Arguments

`z`	numeric or complex vector
`s`	complex number; current implementation is aimed at $s \in \{0, -1, \dots\}$
`method`	a string specifying the algorithm to be used.
`logarithm`	logical specified to return log(Li.(.)) instead of Li.(.)
`is.log.z`	logical; if TRUE, the specified `z` argument is really $w = \log(z)$; that is, we compute $\mathrm{Li}_s(\exp(w))$, and we typically have $w < 0$, or equivalently, $z < 1$.
`is.logmlog`	logical; if TRUE, the specified argument `z` is $lw = \log(-w) = \log(-\log(z))$ (where as above, $w = \log(z)$).

asymp.w.order	currently only default is implemented.
n.sum	for method="sum" only: the number of terms used.
x	numeric vector, may contain Inf, NA, and negative values.

Details

Almost entirely taken from http://en.wikipedia.org/wiki/Polylogarithm:

For integer values of the polylogarithm order, the following explicit expressions are obtained by repeated application of $z \frac{\partial}{\partial z}$ to $\mathrm{Li}_1(z)$:

$$\mathrm{Li}_1(z) = -\log(1-z), \;\; \mathrm{Li}_0(z) = \frac{z}{1-z}, \;\; \mathrm{Li}_{-1}(z) = \frac{z}{(1-z)^2}, \;\; \mathrm{Li}_{-2}(z) = \frac{z\,(1+z)}{(1-z)^3},$$

$\mathrm{Li}_{-3}(z) = \frac{z\,(1+4z+z^2)}{(1-z)^4}$, etc.

Accordingly, the polylogarithm reduces to a ratio of polynomials in z, and is therefore a rational function of z, for all nonpositive integer orders. The general case may be expressed as a finite sum:

$$\mathrm{Li}_{-n}(z) = \left(z \frac{\partial}{\partial z}\right)^n \frac{z}{1-z} = \sum_{k=0}^{n} k!\, S(n+1, k+1) \left(\frac{z}{1-z}\right)^{k+1} \quad (n = 0, 1, 2, \ldots),$$

where $S(n,k)$ are the Stirling numbers of the second kind.

Equivalent formulae applicable to negative integer orders are (Wood 1992, § 6) ...

$$\mathrm{Li}_{-n}(z) = \frac{1}{(1-z)^{n+1}} \sum_{k=0}^{n-1} \left\langle {n \atop k} \right\rangle z^{n-k} = \frac{z \sum_{k=0}^{n-1} \left\langle {n \atop k} \right\rangle z^k}{(1-z)^{n+1}}, \qquad (n = 1, 2, 3, \ldots),$$

where $\left\langle {n \atop k} \right\rangle$ are the Eulerian numbers; see also Eulerian.

Value

numeric/complex vector as z, or x, respectively.

References

Wikipedia (2011) *Polylogarithm*, http://en.wikipedia.org/wiki/Polylogarithm.

Wood, D. C. (June 1992). The Computation of Polylogarithms. Technical Report 15-92. Canterbury, UK: University of Kent Computing Laboratory. http://www.cs.kent.ac.uk/pubs/1992/110.

Apostol, T. M. (2010), *"Polylogarithm"*, in the NIST Handbook of Mathematical Functions, http://dlmf.nist.gov/25.12

Lewin, L. (1981). *Polylogarithms and Associated Functions*. New York: North-Holland. ISBN 0-444-00550-1.

For Debye functions: Levin (1981) above, and
Wikipedia (2014) *Debye function*, http://en.wikipedia.org/wiki/Debye_function.

See Also

The polylogarithm is used in MLE for some Archimedean copulas; see `emle`;

The Debye functions are used for `tau` or `rho` computations of the Frank copula.

Examples

```
## The dilogarithm,  polylog(z, s = 2) = Li_2(.) -- mathmatically defined on C \ [1, Inf)
## so x -> 1 is a limit case:
polylog(z = 1, s = 2)
## in the limit, should be equal to
pi^2 / 6

## Default method uses  GSL's dilog():
rLi2 <- curve(polylog(x, 2), -5, 1, n= 1+ 6*64, col=2, lwd=2)
abline(c(0,1), h=0,v=0:1, lty=3, col="gray40")
## "sum" method gives the same for |z| < 1 and large number of terms:
ii <- which(abs(rLi2$x) < 1)
stopifnot(all.equal(rLi2$y[ii],
            polylog(rLi2$x[ii], 2, "sum", n.sum = 1e5),
          tolerance = 1e-15))


z1 <- c(0.95, 0.99, 0.995, 0.999, 0.9999)
L   <- polylog(         z1,  s=-3,method="negI-s-Euler") # close to Inf
LL  <- polylog(     log(z1), s=-3,method="negI-s-Euler",is.log.z=TRUE)
LLL <- polylog(log(-log(z1)),s=-3,method="negI-s-Euler",is.logmlog=TRUE)
all.equal(L, LL)
all.equal(L, LLL)

p.Li <- function(s.set, from = -2.6, to = 1/4, ylim = c(-1, 0.5),
                 colors = c("orange","brown", palette()), n = 201, ...)
{
    s.set <- sort(s.set, decreasing = TRUE)
    s <- s.set[1] # <_ for auto-ylab
    curve(polylog(x, s, method="negI-s-Stirling"), from, to,
          col=colors[1], ylim=ylim, n=n, ...)
    abline(h=0,v=0, col="gray")
    for(is in seq_along(s.set)[-1])
        curve(polylog(x, s=s.set[is], method="negI-s-Stirling"),
              add=TRUE, col = colors[is], n=n)
    s <- rev(s.set)
    legend("bottomright",paste("s =",s), col=colors[2-s], lty=1, bty="n")
}

## yellow is unbearable (on white):
palette(local({p <- palette(); p[p=="yellow"] <- "goldenrod"; p}))

## Wikipedia page plot (+/-):
p.Li(1:-3, ylim= c(-.8, 0.6), colors = c(2:4,6:7))

## and a bit more:
p.Li(1:-5)
```

```
## For the range we need it:
ccol <- c(NA,NA, rep(palette(),10))
p.Li(-1:-20, from=0, to=.99, colors=ccol, ylim = c(0, 10))
## log-y scale:
p.Li(-1:-20, from=0, to=.99, colors=ccol, ylim = c(.01, 1e7),
     log = "y", yaxt = "n")
if(require(sfsmisc)) eaxis(2) else axis(2)
```

`polynEval` *Evaluate Polynomials*

Description

Evaluate a univariate polynomial at `x` (typically a vector), that is, compute, for a given vector of coefficients `coef`, the polynomial `coef[1] + coef[2]*x + ... + coef[p+1]*x^p`.

Usage

```
polynEval(coef, x)
```

Arguments

`coef`	numeric vector. If a vector, `x` can be an array and the result matches `x`.
`x`	numeric vector or array.

Details

The stable Horner rule is used for evaluation.

Using the C code speeds up the already fast R code available in `polyn.eval()` in package **sfsmisc**.

Value

numeric vector or array, with the same dimensions as `x`, containing the polynomial values $p(x)$.

See Also

For a much more sophisticated treatment of polynomials, use the `polynom` package (for example, evaluation can be done via `predict.polynomial`).

Examples

```
polynEval(c(1,-2,1), x = -2:7) # (x - 1)^2
polynEval(c(0, 24, -50, 35, -10, 1),
          x = matrix(0:5, 2,3)) # 5 zeros!
```

`printNacopula` *Print Compact Overview of a Nested Archimedean Copula ("nacopula")*

Description

Print a compact overview of a nested Archimedean copula, that is, an object of class "nacopula". Calling `printNacopula` explicitly allows to customize the printing behavior. Otherwise, the `show()` method calls `printNacopula` with default arguments only.

Usage

```
printNacopula(x, labelKids=NA, deltaInd=, indent.str="",
              digits=getOption("digits"),
              width=getOption("width"), ...)
```

Arguments

`x`	an R object of class `nacopula`.
`labelKids`	logical specifying if child copulas should be labeled; If `NA` (as per default), on each level, children are labeled only if they are not only-child.
`deltaInd`	by how much should each child be indented *more* than its parent? (non-negative integer). The default is three with `labelKids` being the default or `TRUE`, otherwise it is five (for `labelKids=FALSE`).
`indent.str`	a `character` string specifying the indentation, that is, the string that should be *prepended* on the first line of output, and determine the amount of blanks for the remaining lines.
`digits, width`	number of significant digits, and desired print width; see `print.default`.
`...`	potentially further arguments, passed to methods.

Value

invisibly, `x`.

Examples

```
C8 <- onacopula("F", C(1.9, 1,
                       list(K1 = C(5.7, c(2,5)),
                            abc= C(5.0, c(3,4,6),
                                   list(L2 = C(11.5, 7:8))))))
C8 # -> printNacopula(C8)
printNacopula(C8, delta=10)
printNacopula(C8, labelKids=TRUE)
```

prob *Computing Probabilities of Hypercubes*

Description

Compute probabilities of a $d-$dimensional random vector U distributed according to a given copula x to fall in a hypercube $(l, u]$, where l and u denote the lower and upper corners of the hypercube, respectively.

Usage

```
prob(x, l, u)
```

Arguments

x	copula of dimension d, that is, an object inheriting from Copula.
l, u	d-dimensional, numeric, lower and upper hypercube boundaries, respectively, satisfying $0 \le l_i \le u_i \le 1$, for $i \in 1, \ldots, d$.

Value

A numeric in $[0, 1]$ which is the probability $P(l_i < U_i \le u_i)$.

See Also

pCopula(.).

Examples

```
## Construct a three-dimensional nested Joe copula with parameters
## chosen such that the Kendall's tau of the respective bivariate margins
## are 0.2 and 0.5.
theta0 <- copJoe@iTau(.2)
theta1 <- copJoe@iTau(.5)
C3 <- onacopula("J", C(theta0, 1, C(theta1, c(2,3))))

## Compute the probability of a random vector distributed according to
## this copula to fall inside the cube with lower point l and upper
## point u.
l <- c(.7,.8,.6)
u <- c(1,1,1)
prob(C3, l, u)

## ditto for a bivariate normal copula with rho = 0.8 :
prob(normalCopula(0.8), c(.2,.4), c(.3,.6))
```

qqplot2 *Q-Q Plot with Rugs and Pointwise Asymptotic Confidence Intervals*

Description

A Q-Q plot (possibly) with rugs and pointwise approximate (via the Central Limit Theorem) two-sided 1-α confidence intervals.

Usage

```
qqplot2(x, qF, log = "", qqline.args = if(log=="x" || log=="y")
        list(untf=TRUE) else list(),
        rug.args = list(tcl=-0.6*par("tcl")),
        alpha = 0.05, CI.args = list(col="gray40"),
        CI.mtext = list(text=paste0("Pointwise asymptotic ", 100*(1-alpha),
                        "% confidence intervals"), side=4,
                        cex=0.6*par("cex.main"), adj=0, col="gray40"),
        main = quote(bold(italic(F)~~"Q-Q plot")),
        main.args = list(text=main, side=3, line=1.1, cex=par("cex.main"),
                         font=par("font.main"), adj=par("adj"), xpd=NA),
        xlab = "Theoretical quantiles", ylab = "Sample quantiles",
        file="", width=6, height=6, ...)
```

Arguments

x	numeric.
qF	(theoretical) quantile function against which the Q-Q plot is created.
log	character string indicating whether log-scale should be used; see ?plot.default.
qqline.args	argument list passed to qqline() for creating the Q-Q line. Use qqline.args=NULL to omit the Q-Q line.
rug.args	argument list passed to rug() for creating the rugs. Use rug.args=NULL to omit the rugs.
alpha	significance level.
CI.args	argument list passed to lines() for plotting the confidence intervals. Use CI.args=NULL to omit the confidence intervals.
CI.mtext	argument list passed to mtext() for plotting information about the confidence intervals. Use CI.mtext=NULL to omit the information.
main	title (can be an expression; use "" for no title).
main.args	argument list passed to mtext() for plotting the title. Use main.args=NULL to omit the title.
xlab	x axis label.
ylab	y axis label.
file	file name including the extension ".pdf".

`width`	width parameter of `pdf()`.
`height`	height parameter of `pdf()`.
`...`	additional arguments passed to `plot()` based for plotting the points.

Details

See the source code for how the confidence intervals are constructed precisely.

Value

`invisible()`.

See Also

`plot()` for the underlying plot function, `qqline()` for how the Q-Q line is implemented, `rug()` for how the rugs are constructed, `lines()` for how the confidence intervals are drawn, and `mtext()` for how the title and information about the confidence intervals is printed. `pdf()` for plotting to pdf.

Examples

```
n <- 250
df <- 7
set.seed(1)
x <- rchisq(n, df=df)

## Q-Q plot against the true quantiles (of a chi^2_3 distribution)
qqplot2(x, qF = function(p) qchisq(p, df=df),
        main = substitute(bold(italic(chi[NU])~~"Q-Q Plot"), list(NU=df)))

## in log-log scale
qqplot2(x, qF = function(p) qchisq(p, df=df), log="xy",
        main = substitute(bold(italic(chi[NU])~~"Q-Q Plot"), list(NU=df)))

## Q-Q plot against wrong quantiles (of an Exp(1) distribution)
qqplot2(x, qF=qexp, main = quote(bold(Exp(1)~~"Q-Q Plot")))
```

`radSymTest` *Test of Exchangeability for a Bivariate Copula*

Description

Test for assessing the radial symmetry of the underlying multivariate copula based on the empirical copula. The test statistic is a multivariate extension of the definition adopted in the first reference. An approximate p-value for the test statistic is obtained by means of a appropriate *bootstrap* which can take the presence of ties in the component series of the data into account; see the second reference.

Usage

```
radSymTest(x, N = 1000, ties = NA)
```

Arguments

`x`	a data matrix that will be transformed to pseudo-observations.
`N`	number of boostrap iterations to be used to simulate realizations of the test statistic under the null hypothesis.
`ties`	logical; if `TRUE`, the boostrap procedure is adapted to the presence of ties in any of the coordinate samples of `x`; the default value of `NA` indicates that the presence/absence of ties will be checked for automatically.

Details

More details are available in the second reference.

Value

An object of `class` `htest` which is a list, some of the components of which are

`statistic`	value of the test statistic.
`p.value`	corresponding approximate p-value.

References

Genest, C. and G. Nešlehová, J. (2014). On tests of radial symmetry for bivariate copulas. *Statistical Papers* **55**, 1107–1119.

Kojadinovic, I. (2017). Some copula inference procedures adapted to the presence of ties. *Computational Statistics and Data Analysis* **112**, 24–41, http://arxiv.org/abs/1609.05519.

See Also

`exchTest`, `exchEVTest`, `gofCopula`.

Examples

```
## Data from radially symmetric copulas
radSymTest(rCopula(200, frankCopula(3)))
radSymTest(rCopula(200, normalCopula(0.7, dim = 3)))

## Data from non radially symmetric copulas
radSymTest(rCopula(200, claytonCopula(3)))
radSymTest(rCopula(200, gumbelCopula(2, dim=3)))
```

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

Description

Five years of daily log-returns (from 1996 to 2000) of 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` the date, of class "`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

McNeil, A. J., Frey, R., and Embrechts, P. (2005). *Quantitative Risk Management: Concepts, Techniques, Tools*. Princeton University Press.

Examples

```
data(rdj)
str(rdj)# 'Date' is of class "Date"

with(rdj, {
   matplot(Date, rdj[,-1], type = "o", xaxt = "n", ylim = .15* c(-1,1),
           main = paste("rdj - data;  n =", nrow(rdj)))
   Axis(Date, side=1)
})
legend("top", paste(1:3, names(rdj[,-1])), col=1:3, lty=1:3, bty="n")


x <- rdj[, -1] # '-1' : not the Date
## a t-copula (with a vague inital guess of Rho entries)
tCop <- tCopula(rep(.2, 3), dim=3, dispstr="un", df=10, df.fixed=TRUE)
ft <- fitCopula(tCop, data = pobs(x))
ft
ft@copula # the fitted t-copula as tCopula object
system.time(
g.C <- gofCopula(claytonCopula(dim=3), as.matrix(x), simulation = "mult")
) ## 5.3 sec
```

```
system.time(
g.t <- gofCopula(ft@copula, as.matrix(x), simulation = "mult")
) ## 8.1 sec
```

---

retstable *Sampling Exponentially Tilted Stable Distributions*

Description

Generating random variates of an exponentially tilted stable distribution of the form

$$\tilde{S}(\alpha, 1, (\cos(\alpha\pi/2)V_0)^{1/\alpha}, V_0\mathbf{1}_{\{\alpha=1\}}, h\mathbf{1}_{\{\alpha\neq 1\}}; 1),$$

with parameters $\alpha \in (0,1]$, $V_0 \in (0,\infty)$, and $h \in [0,\infty)$ and corresponding Laplace-Stieltjes transform

$$\exp(-V_0((h+t)^\alpha - h^\alpha)),\ t \in [0,\infty];$$

see the references for more details about this distribution.

Usage

```
retstable(alpha, V0, h = 1, method = NULL)
retstableR(alpha, V0, h = 1)
```

Arguments

alpha	parameter in $(0,1]$.
V0	vector of values in $(0,\infty)$ (for example, when sampling nested Clayton copulas, these are random variates from F_0), that is, the distribution corresponding to ψ_0.
h	parameter in $[0,\infty)$.
method	a character string denoting the method to use, currently either "MH" (Marius Hofert's algorithm) or "LD" (Luc Devroye's algorithm). By default, when NULL, a smart choice is made to use the fastest of these methods depending on the specific values of V_0.

Details

retstableR is a pure R version of "MH", however, not as fast as retstable (implemented in C, based on both methods) and therefore not recommended in simulations when run time matters.

Value

A vector of variates from $\tilde{S}(\alpha, 1, ...)$; see above.

References

Devroye, L. (2009) Random variate generation for exponentially and polynomially tilted stable distributions, *ACM Transactions on Modeling and Computer Simulation* **19**, 18, 1–20.

Hofert, M. (2011) Efficiently sampling nested Archimedean copulas, *Computational Statistics & Data Analysis* **55**, 57–70.

Hofert, M. (2012), Sampling exponentially tilted stable distributions, *ACM Transactions on Modeling and Computer Simulation* **22**, 1.

See Also

`rstable1` for sampling stable distributions.

Examples

```
## Draw random variates from an exponentially tilted stable distribution
## with given alpha, V0, and h = 1
alpha <- .2
V0 <- rgamma(200, 1)
rETS <- retstable(alpha, V0)

## Distribution plot the random variates -- log-scaled
hist(log(rETS), prob=TRUE)
lines(density(log(rETS)), col=2)
rug (log(rETS))
```

---

rF01FrankJoe — *Sample Univariate Distributions Involved in Nested Frank and Joe Copulas*

Description

`rF01Frank`: Generate a vector of random variates $V_{01} \sim F_{01}$ with Laplace-Stieltjes transform

$$\psi_{01}(t; V_0) = \Big(\frac{1-(1-\exp(-t)(1-e^{-\theta_1}))^{\theta_0/\theta_1}}{1-e^{-\theta_0}}\Big)^{V_0}.$$

for the given realizations V_0 of Frank's F_0 and the parameters $\theta_0, \theta_1 \in (0,\infty)$ such that $\theta_0 \le \theta_1$. This distribution appears on sampling nested Frank copulas. The parameter `rej` is used to determine the cut-off point of two algorithms that are involved in sampling F_{01}. If `rej` $< V_0\theta_0(1-e^{-\theta_0})^{V_0-1}$ a rejection from F_{01} of Joe is applied (see `rF01Joe`; the meaning of the parameter `approx` is explained below), otherwise a sum is sampled with a logarithmic envelope for each summand.

`rF01Joe`: Generate a vector of random variates $V_{01} \sim F_{01}$ with Laplace-Stieltjes transform

$$\psi_{01}(t; V_0) = (1-(1-\exp(-t))^{\alpha})^{V_0}.$$

for the given realizations V_0 of Joe's F_0 and the parameter $\alpha \in (0,1]$. This distribution appears on sampling nested Joe copulas. Here, $\alpha = \theta_0/\theta_1$, where $\theta_0, \theta_1 \in [1,\infty)$ such that $\theta_0 \le \theta_1$. The parameter approx denotes the largest number of summands in the sum-representation of V_{01} before the asymptotic

$$V_{01} = V_0^{1/\alpha} S(\alpha, 1, \cos^{1/\alpha}(\alpha\pi/2), \mathbf{1}_{\{\alpha=1\}}; 1)$$

is used to sample V_{01}.

Usage

```
rF01Frank(V0, theta0, theta1, rej, approx)
rF01Joe(V0, alpha, approx)
```

Arguments

V0	a vector of random variates from F_0.
theta0, theta1, alpha	parameters θ_0, θ_1 and α as described above.
rej	parameter value as described above.
approx	parameter value as described above.

Value

A vector of positive integers of length n containing the generated random variates.

References

Hofert, M. (2011). Efficiently sampling nested Archimedean copulas. *Computational Statistics & Data Analysis* **55**, 57–70.

See Also

rFFrank, rFJoe, rSibuya, and rnacopula.

rnacopula

Examples

```
## Sample n random variates V0 ~ F0 for Frank and Joe with parameter
## chosen such that Kendall's tau equals 0.2 and plot histogram
n <- 1000
theta0.F <- copFrank@iTau(0.2)
V0.F <- copFrank@V0(n,theta0.F)
hist(log(V0.F), prob=TRUE); lines(density(log(V0.F)), col=2, lwd=2)
theta0.J <- copJoe@iTau(0.2)
V0.J <- copJoe@V0(n,theta0.J)
hist(log(V0.J), prob=TRUE); lines(density(log(V0.J)), col=2, lwd=2)

## Sample corresponding V01 ~ F01 for Frank and Joe and plot histogram
## copFrank@V01 calls rF01Frank(V0, theta0, theta1, rej=1, approx=10000)
## copJoe@V01 calls rF01Joe(V0, alpha, approx=10000)
theta1.F <- copFrank@iTau(0.5)
V01.F <- copFrank@V01(V0.F,theta0.F,theta1.F)
```

```
hist(log(V01.F), prob=TRUE); lines(density(log(V01.F)), col=2, lwd=2)
theta1.J <- copJoe@iTau(0.5)
V01.J <- copJoe@V01(V0.J,theta0.J,theta1.J)
hist(log(V01.J), prob=TRUE); lines(density(log(V01.J)), col=2, lwd=2)
```

---

rFFrankJoe *Sampling Distribution F for Frank and Joe*

Description

Generate a vector of variates $V \sim F$ from the distribution function F with Laplace-Stieltjes transform

$$(1-(1-\exp(-t)(1-e^{-\theta_1}))^{\alpha})/(1-e^{-\theta_0}),$$

for Frank, or

$$1-(1-\exp(-t))^{\alpha},$$

for Joe, respectively, where θ_0 and θ_1 denote two parameters of Frank (that is, $\theta_0, \theta_1 \in (0,\infty)$) and Joe (that is, $\theta_0, \theta_1 \in [1,\infty)$) satisfying $\theta_0 \le \theta_1$ and $\alpha = \theta_0/\theta_1$.

Usage

```
rFFrank(n, theta0, theta1, rej)
rFJoe(n, alpha)
```

Arguments

n	number of variates from F.
theta0	parameter θ_0.
theta1	parameter θ_1.
rej	method switch for `rFFrank`: if `theta0` > `rej` a rejection from Joe's family (Sibuya distribution) is applied (otherwise, a logarithmic envelope is used).
alpha	parameter $\alpha = \theta_0/\theta_1$ in $(0, 1]$ for `rFJoe`.

Details

`rFFrank(n, theta0, theta1, rej)` calls `rF01Frank(rep(1,n), theta0, theta1, rej, 1)` and `rFJoe(n, alpha)` calls `rSibuya(n, alpha)`.

Value

numeric vector of random variates V of length `n`.

See Also

`rF01Frank`, `rF01Joe`, also for references. `rSibuya`, and `rnacopula`.

Examples

```
## Simple definition of the functions:
rFFrank
rFJoe
```

---

rlog *Sampling Logarithmic Distributions*

Description

Generating random variates from a Log(p) distribution with probability mass function

$$p_k = \frac{p^k}{-\log(1-p)k},\ k \in \mathbf{N},$$

where $p \in (0,1)$. The implemented algorithm is the one named "LK" in Kemp (1981).

Usage

```
rlog(n, p, Ip = 1 - p)
```

Arguments

n	sample size, that is, length of the resulting vector of random variates.
p	parameter in $(0,1)$.
Ip	$= 1 - p$, possibly more accurate, e.g, when $p \approx 1$.

Details

For documentation and didactical purposes, `rlogR` is a pure-R implementation of `rlog`. However, `rlogR` is not as fast as `rlog` (the latter being implemented in C).

Value

A vector of positive `integer`s of length `n` containing the generated random variates.

References

Kemp, A. W. (1981), Efficient Generation of Logarithmically Distributed Pseudo-Random Variables, *Journal of the Royal Statistical Society: Series C (Applied Statistics)* **30**, 3, 249–253.

Examples

```
## Sample n random variates from a Log(p) distribution and plot a
## histogram
n <- 1000
p <- .5
X <- rlog(n, p)
hist(X, prob = TRUE)
```

`rnacModel` *Random nacopula Model*

Description

Randomly construct a nested Archimedean copula model,

Usage

```
rnacModel(family, d, pr.comp, rtau0 = function() rbeta(1, 2,4),
          order=c("random", "each", "seq"), digits.theta = 2)
```

Arguments

`family`	the Archimedean family
`d`	integer >=2; the dimension
`pr.comp`	probability of a direct component on each level
`rtau0`	a `function` to generate a (random) tau, corresponding to theta0, the outermost theta.
`order`	string indicating how the component IDs are selected.
`digits.theta`	integer specifying the number of digits to round the theta values.

Value

an object of `outer_nacopula`.

See Also

`rnacopula` for generating d-dimensional observations from an (outer) `nacopula`, e.g., from the *result* of `rnacModel()`.

Examples

```
## Implicitly tests the function {with validity of outer_nacopula ..}
set.seed(11)
for(i in 1:40) {
  m1 <- rnacModel("Gumbel", d=sample(20:25, 1), pr.comp = 0.3,
  rtau0 = function() 0.25)
  m2 <- rnacModel("Joe", d=3, pr.comp = 0.1, order="each")
  mC <- rnacModel("Clayton", d=20, pr.comp = 0.3,
  rtau0 = function() runif(1, 0.1, 0.5))
  mF <- rnacModel("Frank", d=sample(20:25, 1), pr.comp = 0.3, order="seq")
}
```

rnacopula *Sampling Nested Archimedean Copulas*

Description

Random number generation for nested Archimedean copulas (of class outer_nacopula, specifically), aka *sampling* nested Archimedean copulas will generate n random vectors of dimension d (= dim(x)).

Usage

```
rnacopula(n, copula, x, ...)
```

Arguments

n	integer specifying the sample size, that is, the number of copula-distributed random vectors $\mathbf{U}_i$, to be generated.
copula	an R object of class "outer_nacopula", typically from onacopula().
x	only for back compatibility: former name of copula argument.
...	possibly further arguments for the given copula family.

Details

The generation happens by calling rnchild() on each child copula (which itself recursively descends the tree implied by the nested Archimedean structure). The algorithm is based on a mixture representation of the generic distribution functions F_0 and F_{01} and is presented in McNeil (2008) and Hofert (2011a). Details about how to efficiently sample the distribution functions F_0 and F_{01} can be found in Hofert (2010), Hofert (2012), and Hofert and Mächler (2011).

Value

numeric matrix containing the generated vectors of random variates from the nested Archimedean copula object copula.

References

McNeil, A. J. (2008). Sampling nested Archimedean copulas. *Journal of Statistical Computation and Simulation* **78**, 6, 567–581.

Hofert, M. (2010). Efficiently sampling nested Archimedean copulas. *Computational Statistics & Data Analysis* **55**, 57–70.

Hofert, M. (2012), A stochastic representation and sampling algorithm for nested Archimedean copulas. *Journal of Statistical Computation and Simulation*, **82**, 9, 1239–1255.

Hofert, M. (2012). Sampling exponentially tilted stable distributions. *ACM Transactions on Modeling and Computer Simulation* **22**, 1 (3rd article).

Hofert, M. and Mächler, M. (2011). Nested Archimedean Copulas Meet R: The nacopula Package. *Journal of Statistical Software* **39**, 9, 1–20.

See Also

rnchild; classes "nacopula" and "outer_nacopula"; see also onacopula(). rnacModel creates random nacopula *models*, i.e., the input copula for rnacopula(n, copula).

Further, those of the Archimedean families, for example, copGumbel.

Examples

```
## Construct a three-dimensional nested Clayton copula with parameters
## chosen such that the Kendall's tau of the respective bivariate margins
## are 0.2 and 0.5 :
C3 <- onacopula("C", C(copClayton@iTau(0.2), 1,
                       C(copClayton@iTau(0.5), c(2,3))))
C3

## Sample n vectors of random variates from this copula.  This involves
## sampling exponentially tilted stable distributions
n <- 1000
U <- rnacopula(n, C3)

## Plot the drawn vectors of random variates
splom2(U)
```

---

rnchild *Sampling Child 'nacopula's*

Description

Method for generating vectors of random numbers of nested Archimedean copulas which are child copulas.

Usage

```
rnchild(x, theta0, V0, ...)
```

Arguments

x	an "nacopula" object, typically emerging from an "outer_nacopula" object constructed with onacopula().
theta0	the parameter (vector) of the parent Archimedean copula which contains x as a child.
V0	a numeric vector of realizations of V_0 following F_0 whose length determines the number of generated vectors, that is, for each realization V_0, a vector of variates from x is generated.
...	possibly further arguments for the given copula family.

Details

The generation is done recursively, descending the tree implied by the nested Archimedean structure. The algorithm is based on a mixture representation and requires sampling $V_{01} \sim F_{01}$ given random variates $V_0 \sim F_0$. Calling "rnchild" is only intended for experts. The typical call of this function takes place through rnacopula().

Value

a list with components

U	a numeric matrix containing the vector of random variates from the child copula. The number of rows of this matrix therefore equals the length of V_0 and the number of columns corresponds to the dimension of the child copula.

indcol	an integer vector of indices of U (the vector following a nested Archimedean copula of which x is a child) whose corresponding components of U are arguments of the nested Archimedean copula x.

See Also

rnacopula, also for the references. Further, classes "nacopula" and "outer_nacopula"; see also onacopula().

Examples

```
## Construct a three-dimensional nested Clayton copula with parameters
## chosen such that the Kendall's tau of the respective bivariate margins
## are 0.2 and 0.5.
theta0 <- copClayton@iTau(.2)
theta1 <- copClayton@iTau(.5)
C3 <- onacopula("C", C(theta0, 1, C(theta1, c(2,3))))
## Sample n random variates V0 ~ F0 (a Gamma(1/theta0,1) distribution)
n <- 1000
V0 <- copClayton@V0(n, theta0)

## Given these variates V0, sample the child copula, that is, the bivariate
## nested Clayton copula with parameter theta1
U23 <- rnchild(C3@childCops[[1]], theta0, V0)

## Now build the three-dimensional vectors of random variates by hand
U1 <- copClayton@psi(rexp(n)/V0, theta0)
U <- cbind(U1, U23$U)

## Plot the vectors of random variates from the three-dimensional nested
## Clayton copula
splom2(U)
```

rotCopula — *Construction and Class of Rotated aka Reflected Copulas*

Description

Constructs a "reflected" or "rotated" copula from an initial copula and a vector of logicals indicating which dimension to "flip".

Usage

```
rotCopula(copula, flip = TRUE)
```

Arguments

`copula`	an object of class "`Copula`".
`flip`	`logical` vector (of length 1 or `dim(copula)`) indicating which dimension should be "flipped"; by default, all the components are flipped, implying that the resulting copula is the *"survival copula"*.

Value

A "rotated" or "reflected" copula object of class "`rotCopula`".

Slots

of a "`rotCopula`" object

`copula`**:** Object of class "`copula`".

`flip`**:** `logical` vector of length d (the copula dimension) specifying which margins are flipped; corresponds to the `flip` argument of `rotCopula()`.

`dimension`**:** the copula dimension d, an `integer`.

`parameters`**:** `numeric` vector specifying the parameters.

`param.lowbnd`**, and** `param.upbnd`**:** `numeric` vector of the same length as `parameters`, specifying (component wise) bounds for each of the parameters.

`param.names`**:** `character` vector (of same length as `parameters`) with parameter names.

`fullname`**: deprecated**; a `character` string describing the rotated copula.

Note

When there are an even number of flips, the resulting copula can be seen as a *rotated* version of `copula`. In the other cases, e.g., `flip = c(FALSE,TRUE)` in 2d, it is rather a a *reflected* or "mirrored" copula.

See Also

`fitCopula` for fitting such copulas to data, `gofCopula` for goodness-of-fit tests for such copulas.

Examples

```
## Two-dimensional examples: First the Clayton(3) survival copula:
survC <- rotCopula(claytonCopula(3)) # default: flip = 'all TRUE'
contourplot2(survC, dCopula)

## Now, a reflected Clayton copula:
r10C  <- rotCopula(claytonCopula(3), flip = c(TRUE, FALSE))

contourplot2(r10C, dCopula, nlevels = 20, main = "dCopula(<rotCopula>)")
contourplot2(r10C, pCopula, nlevels = 20, main = "pCopula(<rotCopula>)")
rho(r10C)
tau(r10C) # -0.6

n <- 1000
u <- rCopula(n, r10C)
rho.n <- cor(u[,1], u[,2], method = "spearman")
tau.n <- cor(u[,1], u[,2], method = "kendall")

## "Calibration"
rc. <- rotCopula(claytonCopula(), flip = c(TRUE, FALSE))
iRho(rc., rho.n)
iTau(rc., tau.n)

## Fitting
fitCopula(rc., data = pobs(u), method = "irho")
fitCopula(rc., data = pobs(u), method = "itau")
fitCopula(rc., data = pobs(u), method = "mpl")

## Goodness-of-fit testing -- the first, parametric bootstrap, is *really* slow
## Not run: gofCopula(rc., x = u)
gofCopula(rc., x = u, simulation = "mult")

## A four-dimensional example: a rotated Frank copula
rf <- rotCopula(frankCopula(10, dim = 4),
                flip = c(TRUE, FALSE, TRUE, FALSE))

n <- 1000
u <- rCopula(n, rf)
splom2(u)

pCopula(c(0.6,0.7,0.6,0.8), rf)
C.n(cbind(0.6,0.7,0.6,0.8),  u)

## Fitting
(rf. <- rotCopula(frankCopula(dim=4),
                  flip = c(TRUE, FALSE, TRUE, FALSE)))
## fitCopula(rf., data = pobs(u), method = "irho")
## FIXME above: not related to rotCopula but frankCopula
fitCopula(rf., data = pobs(u), method = "itau")
fitCopula(rf., data = pobs(u), method = "mpl")

## Goodness-of-fit testing (first ("PB") is really slow, see above):
```

```
## Not run: gofCopula(rf., x = u)
gofCopula(rf., x = u, simulation = "mult") # takes 3.7 sec [lynne, 2015]
```

RSpobs — *Pseudo-Observations of Radial and Uniform Part of Elliptical and Archimedean Copulas*

Description

Given a matrix of iid multivariate data from a meta-elliptical or meta-Archimedean model, `RSpobs()` computes pseudo-observations of the radial part R and the vector $\boldsymbol{S}$ which follows a uniform distribution on the unit sphere (for elliptical copulas) or the unit simplex (for Archimedean copulas). These quantities can be used for (graphical) goodness-of-fit tests, for example.

Usage

```
RSpobs(x, do.pobs = TRUE, method = c("ellip", "archm"), ...)
```

Arguments

`x`	an (n, d)-`matrix` of data; if `do.pobs=FALSE`, the rows of `x` are assumed to lie in the d-dimensional unit hypercube (if they do not, this leads to an error).
`do.pobs`	`logical` indicating whether `pobs()` is applied to `x` for transforming the data to the d-dimensional unit hypercube.
`method`	`character` string indicating the assumed underlying model, being meta-elliptical if `method="ellip"` (in which case `S` should be approximately uniform on the d-dimensional unit sphere) or meta-Archimedean if `method="archm"` (in which case `S` should be approximately uniform on the d-dimensional unit simplex).
`...`	additional arguments passed to the implemented methods. These can be `method="ellip"` `qQg()` (the quantile `function` of the (assumed) distribution function G_g as given in Genest, Hofert, G. Nešlehová (2014)); if provided, `qQg()` is used in the transformation for obtaining pseudo-observations of R and $\boldsymbol{S}$ (see the code for more details). `method="archm"` `iPsi()` (the assumed underlying generator inverse); if provided, `iPsi()` is used in the transformation for obtaining pseudo-observations of R and $\boldsymbol{S}$ (see the code for more details).

Details

The construction of the pseudo-obersvations of the radial part and the uniform distribution on the unit sphere/simplex is described in Genest, Hofert, G. Nešlehová (2014).

Value

A `list` with components `R` (an n-`vector` containing the pseudo-observations of the radial part) and `S` (an (n, d)-`matrix` containing the pseudo-observations of the uniform distribution (on the unit sphere/simplex)).

References

Genest, C., Hofert, M., G. Nešlehová, J., (2014). Is the dependence Archimedean, elliptical, or what? *To be submitted*.

See Also

`pobs()` for computing the "classical" pseudo-observations.

Examples

```
set.seed(100)
n <- 250 # sample size
d <- 5 # dimension
nu <- 3 # degrees of freedom

## Build a mean vector and a dispersion matrix,
## and generate multivariate t_nu data:
mu <- rev(seq_len(d)) # d, d-1, .., 1
L <- diag(d) # identity in dim d
L[lower.tri(L)] <- 1:(d*(d-1)/2)/d # Cholesky factor (diagonal > 0)
Sigma <- crossprod(L) # pos.-def. dispersion matrix (*not* covariance of X)
X <- rep(mu, each=n) + mvtnorm::rmvt(n, sigma=Sigma, df=nu) # multiv. t_nu data
## note: this is *wrong*: mvtnorm::rmvt(n, mean=mu, sigma=Sigma, df=nu)

## compute pseudo-observations of the radial part and uniform distribution
## once for 1a), once for 1b) below
RS.t    <- RSpobs(X, method="ellip", qQg=function(p) qt(p, df=nu)) # 'correct'
RS.norm <- RSpobs(X, method="ellip", qQg=qnorm) # for testing 'wrong' distribution
stopifnot(length(RS.norm$R) == n, length(RS.t$R) == n,
          dim(RS.norm$S) == c(n,d), dim(RS.t$S) == c(n,d))

## 1) Graphically testing the radial part

## 1a) Q-Q plot of R against the correct quantiles
qqplot2(RS.t$R, qF=function(p) sqrt(d*qf(p, df1=d, df2=nu)),
      main.args=list(text= substitute(bold(italic(F[list(d.,nu.)](r^2/d.))~~"Q-Q Plot"),
                                      list(d.=d, nu.=nu)),
       side=3, cex=1.3, line=1.1, xpd=NA))

## 1b) Q-Q plot of R against the quantiles of F_R for a multivariate normal
##     distribution
qqplot2(RS.norm$R, qF=function(p) sqrt(qchisq(p, df=d)),
       main.args=list(text= substitute(bold(italic(chi[D_]) ~~ "Q-Q Plot"), list(D_=d)),
               side=3, cex=1.3, line=1.1, xpd=NA))

## 2) Graphically testing the angular distribution

## auxiliary function
qqp <- function(k, Bmat) {
    d <- ncol(Bmat) + 1
    qqplot2(Bmat[,k],
            qF = function(p) qbeta(p, shape1=k/2, shape2=(d-k)/2),
```

```
            main.args=list(text= substitute(plain("Beta")(s1,s2) ~~ bold("Q-Q Plot"),
                                        list(s1 = k/2, s2 = (d-k)/2)),
                      side=3, cex=1.3, line=1.1, xpd=NA))
}

## 2a) Q-Q plot of the 'correct' angular distribution
##     (Bmat[,k] should follow a Beta(k/2, (d-k)/2) distribution)
Bmat.t <- gofBTstat(RS.t$S)
qqp(1, Bmat=Bmat.t) # k=1
qqp(3, Bmat=Bmat.t) # k=3

## 2b) Q-Q plot of the 'wrong' angular distribution
Bmat.norm <- gofBTstat(RS.norm$S)
qqp(1, Bmat=Bmat.norm) # k=1
qqp(3, Bmat=Bmat.norm) # k=3

## 3) Graphically check independence between radial part and B_1 and B_3

## 'correct' distributions (multivariate t)
plot(pobs(cbind(RS.t$R, Bmat.t[,1])), # k = 1
          xlab=quote(italic(R)), ylab=quote(italic(B)[1]),
          main=quote(bold("Rank plot between"~~italic(R)~~"and"~~italic(B)[1])))
plot(pobs(cbind(RS.t$R, Bmat.t[,3])), # k = 3
  xlab=quote(italic(R)), ylab=quote(italic(B)[3]),
          main=quote(bold("Rank plot between"~~italic(R)~~"and"~~italic(B)[3])))

## 'wrong' distributions (multivariate normal)
plot(pobs(cbind(RS.norm$R, Bmat.norm[,1])), # k = 1
          xlab=quote(italic(R)), ylab=quote(italic(B)[1]),
          main=quote(bold("Rank plot between"~~italic(R)~~"and"~~italic(B)[1])))
plot(pobs(cbind(RS.norm$R, Bmat.norm[,3])), # k = 3
  xlab=quote(italic(R)), ylab=quote(italic(B)[3]),
          main=quote(bold("Rank plot between"~~italic(R)~~"and"~~italic(B)[3])))
```

`rstable1` *Random numbers from (Skew) Stable Distributions*

Description

Generate random numbers of the stable distribution

$$S(\alpha, \beta, \gamma, \delta; k)$$

with characteristic exponent $\alpha \in (0, 2]$, skewness $\beta \in [-1, 1]$, scale $\gamma \in [0, \infty)$, and location $\delta \in \mathbf{R}$; see Nolan (2010) for the parameterization $k \in \{0, 1\}$. The case $\gamma = 0$ is understood as the unit jump at δ.

Usage

```
rstable1(n, alpha, beta, gamma = 1, delta = 0, pm = 1)
```

Arguments

n	an integer, the number of observations to generate.
alpha	characteristic exponent $\alpha \in (0, 2]$.
beta	skewness $\beta \in [-1, 1]$.
gamma	scale $\gamma \in [0, \infty)$.
delta	location $\delta \in \mathbf{R}$.
pm	0 or 1, denoting which parametrization (as by Nolan) is used.

Details

We use the approach of John Nolan for generating random variates of stable distributions. The function `rstable1` provides two basic parametrizations, by default,

`pm = 1`, the so called "S", "S1", or "1" parameterization. This is the parameterization used by Samorodnitsky and Taqqu (1994), and is a slight modification of Zolotarev's (A) parameterization. It is the form with the most simple form of the characteristic function; see Nolan (2010, p. 8).

`pm = 0` is the "S0" parameterization: based on the (M) representation of Zolotarev for an alpha stable distribution with skewness beta. Unlike the Zolotarev (M) parameterization, gamma and delta are straightforward scale and shift parameters. This representation is continuous in all 4 parameters.

Value

A numeric vector of length `n` containing the generated random variates.

References

Chambers, J. M., Mallows, C. L., and Stuck, B. W. (1976), *A Method for Simulating Stable Random Variables*, J. Amer. Statist. Assoc. **71**, 340–344.

Nolan, J. P. (2012), *Stable Distributions—Models for Heavy Tailed Data*, Birkhaeuser, in progress.

Samoridnitsky, G. and Taqqu, M. S. (1994), *Stable Non-Gaussian Random Processes, Stochastic Models with Infinite Variance*, Chapman and Hall, New York.

See Also

rstable which also allows the 2-parametrization and provides further functionality for stable distributions.

Examples

```
# Generate and plot a series of stable random variates
set.seed(1953)
r <- rstable1(n = 1000, alpha = 1.9, beta = 0.3)
plot(r, type = "l", main = "stable: alpha=1.9 beta=0.3",
     col = "steelblue"); grid()

hist(r, "Scott", prob = TRUE, ylim = c(0,0.3),
     main = "Stable S(1.9, 0.3; 1)")
lines(density(r), col="red2", lwd = 2)
```

safeUroot	*One-dimensional Root (Zero) Finding - Extra "Safety" for Convenience*

Description

`safeUroot()` as a "safe" version of `uniroot()` searches for a root (that is, zero) of the function `f` with respect to its first argument.

"Safe" means searching for the correct `interval = c(lower,upper)` if `sign(f(x))` does not satisfy the requirements at the interval end points; see the 'Details' section.

Usage

```
safeUroot(f, interval, ...,
       lower = min(interval), upper = max(interval),
       f.lower = f(lower, ...), f.upper = f(upper, ...),
       Sig = NULL, check.conv = FALSE,
       tol = .Machine$double.eps^0.25, maxiter = 1000, trace = 0)
```

Arguments

`f`	function
`interval`	interval
`...`	additional named or unnamed arguments to be passed to `f`
`lower, upper`	lower and upper endpoint of search interval
`f.lower, f.upper`	function value at `lower` or `upper` endpoint, respectively.
`Sig`	*desired* sign of `f(upper)`, or `NULL`.
`check.conv`	logical indicating whether a convergence warning of the underlying `uniroot` should be caught as an error.
`tol`	the desired accuracy, that is, convergence tolerance.
`maxiter`	maximal number of iterations
`trace`	number determining tracing

Details

If it is *known how* f changes sign at the root x_0, that is, if the function is increasing or decreasing there, `Sig` can be specified, typically as $S := \pm 1$, to require $S = \mathrm{sign}(f(x_0 + \epsilon))$ at the solution. In that case, the search interval $[l, u]$ must be such that $S * f(l) <= 0$ and $S * f(u) >= 0$.

Otherwise, by default, when `Sig=NULL`, the search interval $[l, u]$ must satisfy $f(l) * f(u) <= 0$.

In both cases, when the requirement is not satisfied, `safeUroot()` tries to enlarge the interval until the requirement *is* satisfied.

Value

A list with four components, `root`, `f.root`, `iter` and `estim.prec`; see `uniroot`.

See Also

`uniroot`.

Examples

```
f1 <- function(x) (121 - x^2)/(x^2+1)
f2 <- function(x) exp(-x)*(x - 12)

try(uniroot(f1, c(0,10)))
try(uniroot(f2, c(0,2)))
##--> error: f() .. end points not of opposite sign

## where as safeUroot() simply first enlarges the search interval:
safeUroot(f1, c(0,10),trace=1)
safeUroot(f2, c(0,2), trace=2)

## no way to find a zero of a positive function:
try( safeUroot(exp, c(0,2), trace=TRUE) )

## Convergence checking :
safeUroot(sinc, c(0,5), maxiter=4) #-> "just" a warning
try( # an error, now with  check.conv=TRUE
  safeUroot(sinc, c(0,5), maxiter=4, check.conv=TRUE) )
```

`serialIndepTest` *Serial Independence Test for Continuous Time Series Via Empirical Copula*

Description

Computes the 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);

(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, verbose = interactive())
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 << lag.max+1` especially when `lag.max` is large.
`N`	number of repetitions when simulating under serial independence.
`verbose`	a logical specifying if progress should be displayed via `txtProgressBar`.
`x`	numeric vector containing the time series whose serial independence is to be tested.
`d`	object of class `serialIndepTestDist` as returned by the function `serialIndepTestSim`. It can be regarded as the empirical distribution of the test statistics under serial independence.
`alpha`	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 S3 class `"serialIndepTestDist"` with list components `sample.size`, `lag.max`, `max.card.subsets`, `number.repetitons`, `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 S3 class `"indepTest"` with list components `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).

The former argument `print.every` is deprecated and not supported anymore; use `verbose` instead.

References

Deheuvels, P. (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.

Deheuvels, P. (1981), A non parametric test for independence, *Publ. Inst. Statist. Univ. Paris*. 26:29–50.

Genest, C. and Rémillard, B. (2004) Tests of independence and randomness based on the empirical copula process. *Test* **13**, 335–369.

Genest, C., Quessy, J.-F., and Rémillard, B. (2006) Local efficiency of a Cramer-von Mises test of independence. *Journal of Multivariate Analysis* **97**, 274–294.

Genest, C., Quessy, J.-F., and Rémillard, B. (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 saveRDS() function can be used.
```

`setTheta` *Specify the Parameter(s) of a Copula*

Description

Set or change the parameter θ (`theta`) of a copula. The name 'theta' has been from its use in (nested) Archimedean copulas, where x is of class "`acopula`" or "`outer_nacopula`". This is used for constructing copula models with specified parameter, as, for example, in `onacopula()`, or also `gofCopula`.

Usage

```
setTheta(x, value, na.ok = TRUE, noCheck = FALSE, freeOnly = TRUE, ...)

## S4 method for signature 'acopula,ANY'
setTheta(x, value, na.ok = TRUE, noCheck = FALSE, freeOnly = TRUE, ...)
## S4 method for signature 'copula,ANY'
setTheta(x, value, na.ok = TRUE, noCheck = FALSE, freeOnly = TRUE, ...)
## S4 method for signature 'xcopula,ANY'
setTheta(x, value, na.ok = TRUE, noCheck = FALSE, freeOnly = TRUE, ...)
## S4 method for signature 'outer_nacopula,numeric'
setTheta(x, value, na.ok = TRUE, noCheck = FALSE, freeOnly = TRUE, ...)
## S4 method for signature 'khoudrajiCopula,ANY'
setTheta(x, value, na.ok = TRUE, noCheck = FALSE, freeOnly = TRUE, ...)

## S4 method for signature 'mixCopula,ANY'
setTheta(x, value, na.ok = TRUE, noCheck = FALSE, freeOnly = TRUE,
         treat.negative = c("set.0", "warn.set0", "stop"), ...)
```

Arguments

`x`	an R object of class `Copula`, i.e., any copula from package **copula**.
`value`	parameter value or vector, `numeric` or `NA` (when `na.ok` is true), corresponding to the *"free"* parameters.
`...`	further arguments for methods.
`na.ok`	logical indicating if `NA` values are ok for `theta`.
`noCheck`	logical indicating if parameter constraint checks should be skipped.
`freeOnly`	logical indicating that only non-fixed aka "free" parameters are to be set. If true as by default, `setTheta()` modifies only the free parameters of the copula; see also `fixParam`.
`treat.negative`	a `character` string indicating how negative mixture weights should be handled. If not "`stop`" which produces an error via `stop`, negative mixture weights are replaced by zero.

Value

an R object of the same class as x, with the main parameter (vector) (often called theta) set to value.

See Also

the "inverse" function, a "getter" method, is getTheta().

Examples

```
myC <- setTheta(copClayton, 0.5)
myC
## Frank copula with Kendall's tau = 0.8 :
(myF.8 <- setTheta(copFrank, iTau(copFrank, tau = 0.8)))

# negative theta is ok for dim = 2 :
myF <- setTheta(copFrank, -2.5, noCheck=TRUE)
myF@tau(myF@theta) # -0.262

myT <- setTheta(tCopula(df.fixed=TRUE), 0.7)
stopifnot(all.equal(myT, tCopula(0.7, df.fixed=TRUE), tolerance=0))

(myT2 <- setTheta(tCopula(dim=3, df.fixed=TRUE), 0.7))
## Setting 'rho' and 'df'  --- for default df.fixed=FALSE :
(myT3 <- setTheta(tCopula(dim=3), c(0.7, 4)))
```

---

show-methods *Methods for 'show()' in Package 'copula'*

Description

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

Methods

object = "parCopula" see Copula.

object = "fitMvdc" (see fitMvdc): and

object = "fitCopula" (see fitCopula): these call the (hidden) print method, with its default argument. Using print() instead, allows to set digits, e.g.

object = "fitMvdc" see fitCopula.

Sibuya *Sibuya Distribution - Sampling and Probabilities*

Description

The Sibuya distribution $\mathrm{Sib}(\alpha)$ can be defined by its Laplace transform

$$1 - (1 - \exp(-t))^{\alpha},\ t \in [0, \infty),$$

its distribution function

$$F(k) = 1 - (-1)^k \binom{\alpha - 1}{k} = 1 - \frac{1}{kB(k, 1-\alpha)},\ k \in \mathbf{N}$$

(where B denotes the beta function) or its probability mass function

$$p_k = \binom{\alpha}{k}(-1)^{k-1},\ k \in \mathbf{N},$$

where $\alpha \in (0, 1]$.

pSibuya evaluates the distribution function.

dSibuya evaluates the probability mass function.

rSibuya generates random variates from $\mathrm{Sib}(\alpha)$ with the algorithm described in Hofert (2011), Proposition 3.2.

dsumSibuya gives the probability mass function of the n-fold convolution of Sibuya variables, that is, the sum of n independent Sibuya random variables, $S = \sum_{i=1}^{n} X_i$, where $X_i \sim \mathrm{Sib}(\alpha)$.

This probability mass function can be shown (see Hofert (2010, pp. 99)) to be

$$\sum_{j=1}^{n} \binom{n}{j}\binom{j\alpha}{k}(-1)^{k-j},\ k \in \{n, n+1, \dots\}.$$

Usage

```
rSibuya(n, alpha)
dSibuya(x, alpha, log=FALSE)
pSibuya(x, alpha, lower.tail=TRUE, log.p=FALSE)

dsumSibuya(x, n, alpha,
           method=c("log", "direct", "diff", "exp.log",
                    "Rmpfr", "Rmpfr0", "RmpfrM", "Rmpfr0M"),
           mpfr.ctrl = list(minPrec = 21, fac = 1.25, verbose=TRUE),
           log=FALSE)
```

Arguments

`n`	for `rSibuya`: sample size, that is, length of the resulting vector of random variates. for `dsumSibuya`: the number n of summands.
`alpha`	parameter in $(0, 1]$.
`x`	vector of `integer` values ("quantiles") x at which to compute the probability mass or cumulative probability.
`log, log.p`	`logical`; if TRUE, probabilities p are given as log(p).
`lower.tail`	`logical`; if TRUE (the default), probabilities are $P(X \le x)$, otherwise, $P(X > x)$.
`method`	character string specifying which computational method is to be applied. Implemented are:

`"log"` evaluates the logarithm of the sum

$$\sum_{j=1}^{n} \binom{n}{j} \binom{j\alpha}{x} (-1)^{x-j}$$

in a numerically stable way;

`"direct"` directly evaluates the sum;

`"Rmpfr*"` are as `method="direct"` but use high-precision arithmetic; `"Rmpfr"` and `"Rmpfr0"` return `double`s whereas `"RmpfrM"` and `"Rmpfr0M"` give `mpfr` high-precision numbers. Whereas `"Rmpfr"` and `"RmpfrM"` each adapt to high enough precision, the `"Rmpfr0*"` ones do not adapt.

For all `"Rmpfr*"` methods, `alpha` can be set to a `mpfr` number of specified precision and this will determine the precision of all parts of the internal computations.

`"diff"` interprets the sum as a forward difference and computes it via `diff`;

`"exp.log"` is as `method="log"` but without numerically stable evaluation (not recommended, use with care).

`mpfr.ctrl`	for `method = "Rmpfr"` or `"RmpfrM"` only: a list of `minPrec`: minimal (estimated) precision in bits, `fac`: factor with which current precision is multiplied if it is not sufficient. `verbose`: determining if and how much is printed.

Details

The Sibuya distribution has **no** finite moments, that is, specifically infinite mean and variance.

For documentation and didactical purposes, `rSibuyaR` is a pure-R implementation of `rSibuya`, of course slower than `rSibuya` as the latter is implemented in C.

Note that the sum to evaluate for `dsumSibuya` is numerically highly challenging, even already for small α values (for example, $n \ge 10$), and therefore should be used with care. It may require high-precision arithmetic which can be accessed with `method="Rmpfr"` (and the **Rmpfr** package).

Value

rSibuya: A vector of positive `integer`s of length `n` containing the generated random variates.

dSibuya, pSibuya: a vector of probabilities of the same length as `x`.

dsumSibuya: a vector of probabilities, positive if and only if `x >= n` and of the same length as `x` (or `n` if that is longer).

References

Hofert, M. (2010). *Sampling Nested Archimedean Copulas with Applications to CDO Pricing*. Südwestdeutscher Verlag fuer Hochschulschriften AG & Co. KG.

Hofert, M. (2011). Efficiently sampling nested Archimedean copulas. *Computational Statistics & Data Analysis* **55**, 57–70.

See Also

`rFJoe` and `rF01Joe` (where `rSibuya` is applied).

Examples

```
## Sample n random variates from a Sibuya(alpha) distribution and plot a
## histogram
n <- 1000
alpha <- .4
X <- rSibuya(n, alpha)
hist(log(X), prob=TRUE); lines(density(log(X)), col=2, lwd=2)
```

`SMI.12` *SMI Data – 141 Days in Winter 2011/2012*

Description

`SMI.12` contains the close prices of all 20 constituents of the Swiss Market Index (SMI) from 2011-09-09 to 2012-03-28.

Usage

```
data(SMI.12)
```

Format

`SMI.12` is conceptually a multivariate time series, here simply stored as `numeric` `matrix`, where the `rownames` are dates (of week days).

The format is:

num [1:141, 1:20] 16.1 15.7 15.7 16.1 16.6 ... - attr(*, "dimnames")=List of 2 ..$: chr [1:141] "2011-09-09" "2011-09-12" "2011-09-13" "2011-09-14" ... ..$: chr [1:20] "ABBN" "ATLN" "ADEN" "CSGN" ...

... from 2011-09-09 to 2012-03-28

`lSMI` is the list of the original data (*before* `NA` "imputation").

Source

The data was drawn from Yahoo! Finance.

Examples

```
data(SMI.12)
## maybe
head(SMI.12)

str(D.12 <- as.Date(rownames(SMI.12)))
summary(D.12)

matplot(D.12, SMI.12, type="l", log = "y",
        main = "The 20 SMI constituents  (2011-09 -- 2012-03)",
        xaxt="n", xlab = "2011  /  2012")
Axis(D, side=1)



if(FALSE) { ##--- This worked up to mid 2012, but no longer ---
 begSMI <- "2011-09-09"
 endSMI <- "2012-03-28"
 ##-- read *public* data ------------------------------
 stopifnot(require(zoo), # -> to access all the zoo methods
           require(tseries))
 symSMI <- c("ABBN.VX","ATLN.VX","ADEN.VX","CSGN.VX","GIVN.VX","HOLN.VX",
     "BAER.VX","NESN.VX","NOVN.VX","CFR.VX", "ROG.VX", "SGSN.VX",
     "UHR.VX", "SREN.VX","SCMN.VX","SYNN.VX","SYST.VX","RIGN.VX",
     "UBSN.VX","ZURN.VX")
 lSMI <- sapply(symSMI, function(sym)
get.hist.quote(instrument = sym, start= begSMI, end= endSMI,
       quote = "Close", provider = "yahoo",
       drop=TRUE))
 ## check if stock data have the same length for each company.
 sapply(lSMI, length)
 ## "concatenate" all:
 SMIo <- do.call(cbind, lSMI)
 ## and fill in the NAs :
 SMI.12 <- na.fill(SMIo, "extend")
 colnames(SMI.12) <- sub("\\.VX", "", colnames(SMI.12))
 SMI.12 <- as.matrix(SMI.12)
}##----       --- original download

if(require(zoo)) {
  stopifnot(identical(SMI.12,
     local({ S <- as.matrix(na.fill(do.call(cbind, lSMI), "extend"))
             colnames(S) <- sub("\\.VX", "", colnames(S)); S })))
}
```

splom2-methods *Methods for Scatter Plot Matrix 'splom2' in Package 'copula'*

Description

Methods `splom2()` to draw scatter-plot matrices of (random samples of) distributions from package **copula**.

Usage

```
## S4 method for signature 'matrix'
splom2(x, varnames = NULL, varnames.null.lab = "U",
      xlab = "", col.mat = NULL, bg.col.mat = NULL, ...)
## ditto an identical  'data.frame'  method

## S4 method for signature 'Copula'
splom2(x, n, ...)
## S4 method for signature 'mvdc'
splom2(x, n, varnames.null.lab = "X", ...)
```

Arguments

`x`	a "`matrix`", "`data.frame`", "`Copula`" or a "`mvdc`" object.
`n`	when `x` is not matrix-like: The sample size of the random sample drawn from `x`.
`varnames`	the variable names, typically unspecified.
`varnames.null.lab`	the `character` string determining the "base name" of the variable labels in case `varnames` is `NULL` and `x` does not have all column names given.
`xlab`	the x-axis label.
`col.mat`	a matrix of colors (or one color) for the plot symbols; if `NULL` (as by default), `trellis.par.get("plot.symbol")$col` is used for all symbols. (Note that in **copula** version 0.999-15, this was not true; instead `"black"` was used.)
`bg.col.mat`	a matrix of colors for the background (the default is the setting as obtained from `trellis.par.get("background")$col`).
`...`	additional arguments passed to the underlying `splom()`.

Value

From `splom()`, an R object of class `"trellis"`.

See Also

`pairs2()` for a similar function (for matrices and data frames) based on `pairs()`.

The **lattice**-based `cloud2-methods` for 3D data, and `wireframe2-methods` and `contourplot2-methods` for functions.

Examples

```
## For 'matrix' objects
## Create a 100 x 7 matrix of random variates from a t distribution
## with four degrees of freedom and plot the generated data
n <- 1000 # sample size
d <- 3 # dimension
nu <- 4 # degrees of freedom
tau <- 0.5 # Kendall's tau
th <- iTau(tCopula(df = nu), tau) # corresponding parameter
cop <- tCopula(th, dim = d, df = nu) # define copula object
set.seed(271)
U <- rCopula(n, copula = cop)
splom2(U)

## For 'copula' objects
set.seed(271)
splom2(cop, n = n) # same as above

## For 'rotCopula' objects: ---> Examples in rotCopula

## For 'mvdc' objects
mvNN <- mvdc(cop, c("norm", "norm", "exp"),
             list(list(mean = 0, sd = 1), list(mean = 1), list(rate = 2)))
splom2(mvNN, n = n)
```

`Stirling` *Eulerian and Stirling Numbers of First and Second Kind*

Description

Compute Eulerian numbers and Stirling numbers of the first and second kind, possibly vectorized for all k “at once”.

Usage

```
Stirling1(n, k)
Stirling2(n, k, method = c("lookup.or.store", "direct"))
Eulerian (n, k, method = c("lookup.or.store", "direct"))

Stirling1.all(n)
Stirling2.all(n)
Eulerian.all (n)
```

Arguments

`n`	positive integer (`0` is allowed for `Eulerian()`).
`k`	integer in `0:n`.

method	for Eulerian() and Stirling2(), string specifying the method to be used. "direct" uses the explicit formula (which may suffer from some cancelation for "large" n).

Details

Eulerian numbers:
$A(n,k) =$ the number of permutations of 1,2,...,n with exactly k ascents (or exactly k descents).

Stirling numbers of the first kind:
s(n,k) = (-1)^n-k times the number of permutations of 1,2,...,n with exactly k cycles.

Stirling numbers of the second kind:
$S_n^{(k)}$ is the number of ways of partitioning a set of n elements into k non-empty subsets.

Value

$A(n,k)$, $s(n,k)$ or $S(n,k) = S_n^{(k)}$, respectively.

Eulerian.all(n) is the same as sapply(0:(n-1), Eulerian, n=n) (for $n > 0$),
Stirling1.all(n) is the same as sapply(1:n, Stirling1, n=n), and
Stirling2.all(n) is the same as sapply(1:n, Stirling2, n=n), but more efficient.

Note

For typical double precision arithmetic,
Eulerian*(n, *) overflow (to Inf) for $n \geq 172$,
Stirling1*(n, *) overflow (to ±Inf) for $n \geq 171$, and
Stirling2*(n, *) overflow (to Inf) for $n \geq 220$.

References

Eulerians:

NIST Digital Library of Mathematical Functions, 26.14: http://dlmf.nist.gov/26.14

Stirling numbers:

Abramowitz and Stegun 24,1,4 (p. 824-5 ; Table 24.4, p.835); Closed Form : p.824 "C."

NIST Digital Library of Mathematical Functions, 26.8: http://dlmf.nist.gov/26.8

Examples

```
Stirling1(7,2)
Stirling2(7,3)

Stirling1.all(9)
Stirling2.all(9)
```

`tauAMH` *Ali-Mikhail-Haq ("AMH")'s and Joe's Kendall's Tau*

Description

Compute Kendall's Tau of an Ali-Mikhail-Haq ("AMH") or Joe Archimedean copula with parameter `theta`. In both cases, analytical expressions are available, but need alternatives in some cases.

`tauAMH()`: Analytically, given as

$$1 - \frac{2((1-\theta)^2 \log(1-\theta) + \theta)}{3\theta^2},$$

for `theta`$= \theta$; numerically, care has to be taken when $\theta \to 0$, avoiding accuracy loss already, for example, for θ as large as `theta = 0.001`.

`tauJoe()`: Analytically,

$$1 - 4\sum_{k=1}^{\infty} \frac{1}{k(\theta k + 2)(\theta(k-1) + 2)},$$

the infinite sum can be expressed by three $\psi()$ (`psigamma`) function terms.

Usage

```
tauAMH(theta)
tauJoe(theta, method = c("hybrid", "digamma", "sum"), noTerms=446)
```

Arguments

`theta`	numeric vector with values in $[-1, 1]$ for AMH, or $[0.238734, Inf)$ for Joe.
`method`	string specifying the method for `tauJoe()`. Use the default, unless for research about the method. Up to **copula** version 0.999-0, the only (implicit) method was `"sum"`.
`noTerms`	the number of summation terms for the `"sum"` method; its default, `446` gives an absolute error smaller than 10^{-5}.

Details

`tauAMH()`: For small `theta` ($= \theta$), we use Taylor series approximations of up to order 7,

$$\tau_A(\theta) = \frac{2}{9}\theta\Big(1 + \theta\Big(\frac{1}{4} + \frac{\theta}{10}\Big(1 + \theta\Big(\frac{1}{2} + \theta\frac{2}{7}\Big)\Big)\Big)\Big) + O(\theta^6),$$

where we found that dropping the last two terms (e.g., only using 5 terms from the $k = 7$ term Taylor polynomial) is actually numerically advantageous.

`tauJoe()`: The `"sum"` method simply replaces the infinite sum by a finite sum (with `noTerms` terms. The more accurate or faster methods, use analytical summation formulas, using the `digamma` aka ψ function, see, e.g., http://en.wikipedia.org/wiki/Digamma_function#Series_formula.

The smallest sensible θ value, i.e., `th` for which `tauJoe(th) == -1` is easily determined via `str(uniroot(function(th) tauJoe(th)-(-1), c(0.1, 0.3), tol = 1e-17), digits=12)` to be `0.2387339899`.

Value

a vector of the same length as `theta` ($= \theta$), with τ values

for `tauAMH`: in $[(5 - 8log2)/3, 1/3]$ $= [-0.1817, 0.3333]$, of $\tau_A(\theta) = 1 - 2(\theta + (1-\theta)^2 \log(1-\theta))/(3\theta^2)$, numerically accurately, to at least around 12 decimal digits.

for `tauJoe`: in [-1,1].

See Also

`acopula-families`, and their class definition, "`acopula`". `etau()` for method-of-moments estimators based on Kendall's tau.

Examples

```
tauAMH(c(0, 2^-40, 2^-20))
curve(tauAMH,  0, 1)
curve(tauAMH, -1, 1)# negative taus as well
curve(tauAMH, 1e-12, 1, log="xy") # linear, tau ~= 2/9*theta in the limit

curve(tauJoe, 1,      10)
curve(tauJoe, 0.2387, 10)# negative taus (*not* valid for Joe: no 2-monotone psi()!)
```

---

uranium — *Uranium Exploration Dataset of Cook & Johnson (1986)*

Description

These data consist of log concentrations of 7 chemical elements in 655 water samples collected near Grand Junction, CO (from the Montrose quad-rangle 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 of the following 7 variables:

`U` (numeric) log concentration of Uranium.

`Li` (numeric) log concentration of Lithium.

`Co` (numeric) log concentration of Colbalt.

`K` (numeric) log concentration of Potassium.

`Cs` (numeric) log concentration of Cesium.

`Sc` (numeric) log concentration of Scandum.

`Ti` (numeric) log concentration of Titanium.

References

Cook, R. D. and Johnson, M. E. (1986) Generalized BurrParetologistic distributions with applications to a uranium exploration data set. *Technometrics* **28**, 123–131.

Examples

```
data(uranium)
```

---

varianceReduction *Variance-Reduction Methods*

Description

Computing antithetic variates or Latin hypercube samples.

Usage

```
rAntitheticVariates(u)
rLatinHypercube(u, ...)
```

Arguments

u	a $n \times d$-matrix (or d-vector) of random variates in the unit hypercube.
...	additional arguments passed to the underlying rank().

Details

rAntitheticVariates() takes any copula sample u, builds $1 - u$, and returns the two matrices in the form of an array; this can be used for the variance-reduction method of (componentwise) antithetic variates.

rLatinHypercube() takes any copula sample, componentwise randomizes its ranks minus 1 and then divides by the sample size in order to obtain a Latin hypercubed sample.

Value

rAntitheticVariates() array of dimension $n \times d \times 2$, say r, where r[,,1] contains the original sample u and r[,,2] contains the sample 1-u.

rLatinHypercube() matrix of the same dimensions as u.

References

Cambou, M., Hofert, M. and Lemieux, C. (2016). Quasi-random numbers for copula models. *Statistics and Computing*, 1–23.

Packham, N. and Schmidt, W. M. (2010). Latin hypercube sampling with dependence and applications in finance. *Journal of Computational Finance* **13**(3), 81–111.

Examples

```
### 1 Basic plots ##############################################################

## Generate data from a Gumbel copula
cop <- gumbelCopula(iTau(gumbelCopula(), tau = 0.5))
n <- 1e4
set.seed(271)
U <- rCopula(n, copula = cop)

## Transform the sample to a Latin Hypercube sample
U.LH <- rLatinHypercube(U)

## Plot
## Note: The 'variance-reducing property' is barely visible, but that's okay
layout(rbind(1:2))
plot(U,    xlab = quote(U[1]), ylab = quote(U[2]), pch = ".", main = "U")
plot(U.LH, xlab = quote(U[1]), ylab = quote(U[2]), pch = ".", main = "U.LH")
layout(1) # reset layout

## Transform the sample to an Antithetic variate sample
U.AV <- rAntitheticVariates(U)
stopifnot(identical(U.AV[,,1],  U ),
          identical(U.AV[,,2], 1-U))

## Plot original sample and its corresponding (componentwise) antithetic variates
layout(rbind(1:2))
plot(U.AV[,,1], xlab = quote(U[1]), ylab = quote(U[2]), pch=".", main= "U")
plot(U.AV[,,2], xlab = quote(U[1]), ylab = quote(U[2]), pch=".", main= "1 - U")
layout(1) # reset layout


### 2 Small variance-reduction study for exceedance probabilities ##############

## Auxiliary function for approximately computing P(U_1 > u_1,..., U_d > u_d)
## by Monte Carlo simulation based on pseudo-random numbers, Latin hypercube
## sampling and quasi-random numbers.
sProb <- function(n, copula, u)
{
    d <- length(u)
    stopifnot(n >= 1, inherits(copula, "Copula"), 0 < u, u < 1,
              d == dim(copula))
    umat <- rep(u, each = n)
    ## Pseudo-random numbers
    U <- rCopula(n, copula = copula)
    PRNG <- mean(rowSums(U > umat) == d)
    ## Latin hypercube sampling (based on the recycled 'U')
    U. <- rLatinHypercube(U)
    LHS <- mean(rowSums(U. > umat) == d)
    ## Quasi-random numbers
    U.. <- cCopula(sobol(n, d = d, randomize = TRUE), copula = copula,
                   inverse = TRUE)
    QRNG <- mean(rowSums(U.. > umat) == d)
```

```
    ## Return
    c(PRNG = PRNG, LHS = LHS, QRNG = QRNG)
}

## Simulate the probabilities of falling in (u_1,1] x ... x (u_d,1]
library(qrng) # for quasi-random numbers
(Xtras <- copula:::doExtras()) # determine whether examples will be extra (long)
B <- if(Xtras)  500 else 100 # number of replications
n <- if(Xtras) 1000 else 200 # sample size
d <- 2 # dimension; note: for d > 2, the true value depends on the seed
nu <- 3 # degrees of freedom
th <- iTau(tCopula(df = nu), tau = 0.5) # correlation parameter
cop <- tCopula(param = th, dim = d, df = nu) # t copula
u <- rep(0.99, d) # lower-left endpoint of the considered cube
set.seed(42) # for reproducibility
true <- prob(cop, l = u, u = rep(1, d)) # true exceedance probability
system.time(res <- replicate(B, sProb(n, copula = cop, u = u)))

## "abbreviations":
PRNG <- res["PRNG",]
LHS  <- res["LHS" ,]
QRNG <- res["QRNG",]

## Compute the variance-reduction factors and % improvements
vrf  <- var(PRNG) / var(LHS)                    # variance reduction factor w.r.t. LHS
vrf. <- var(PRNG) / var(QRNG)                   # variance reduction factor w.r.t. QRNG
pim  <- (var(PRNG) - var(LHS)) / var(PRNG) *100 # improvement w.r.t. LHS
pim. <- (var(PRNG) - var(QRNG))/ var(PRNG) *100 # improvement w.r.t. QRNG

## Boxplot
boxplot(list(PRNG = PRNG, LHS = LHS, QRNG = QRNG), notch = TRUE,
        main = substitute("Simulated exceedance probabilities" ~
                              P(bold(U) > bold(u))~~ "for a" ~ t[nu.]~"copula",
                          list(nu. = nu)),
        sub = sprintf(
          "Variance-reduction factors and %% improvements: %.1f (%.0f%%), %.1f (%.0f%%)",
            vrf, pim, vrf., pim.))
abline(h = true, lty = 3) # true value
mtext(sprintf("B = %d replications with n = %d and d = %d", B, n, d), side = 3)
```

`wireframe2-methods` *Perspective Plots - 'wireframe2' in Package 'copula'*

Description

Generic function and methods `wireframe2()` to draw (**lattice**) `wireframe` (aka "perspective") plots of two-dimensional distributions from package **copula**.

Usage

```
## S4 method for signature 'matrix'
wireframe2(x,
      xlim = range(x[,1], finite = TRUE),
      ylim = range(x[,2], finite = TRUE),
      zlim = range(x[,3], finite = TRUE),
      xlab = NULL, ylab = NULL, zlab = NULL,
      alpha.regions = 0.5, scales = list(arrows = FALSE, col = "black"),
      par.settings = standard.theme(color = FALSE),
      draw.4.pCoplines = FALSE, ...)

## _identical_ method for  'data.frame' instead of 'matrix'

## S4 method for signature 'Copula'
wireframe2(x, FUN, n.grid = 26, delta = 0,
      xlim = 0:1, ylim = 0:1, zlim = NULL,
      xlab = quote(u[1]), ylab = quote(u[2]),
      zlab = list(deparse(substitute(FUN))[1], rot = 90),
      draw.4.pCoplines = identical(FUN, pCopula), ...)

## S4 method for signature 'mvdc'
wireframe2(x, FUN, n.grid = 26, xlim, ylim, zlim = NULL,
      xlab = quote(x[1]), ylab = quote(x[2]),
      zlab = list(deparse(substitute(FUN))[1], rot = 90), ...)
```

Arguments

`x` a "`matrix`", "`data.frame`", "`Copula`" or a "`mvdc`" object.

`xlim, ylim, zlim` the x-, y- and z-axis limits.

`xlab, ylab, zlab` the x-, y- and z-axis labels.

`alpha.regions` see `wireframe`().

`scales` a `list` determining how the axes are drawn; see `wireframe`().

`par.settings` See `wireframe`().

`FUN` the `function` to be plotted; for a "copula", typically `dCopula` or `pCopula`; for an "mvdc", rather `dMvdc`, etc.

`n.grid` the number of grid points used in each dimension. This can be a vector of length two, giving the number of grid points used in x- and y-direction, respectively; the function `FUN` will be evaluated on the corresponding (x,y)-grid.

`delta` a small number in $[0, \frac{1}{2})$ influencing the evaluation boundaries. The x- and y-vectors will have the range `[0+delta, 1-delta]`, the default being `[0,1]`.

`draw.4.pCoplines` logical indicating if the 4 known border segments of a copula distribution function, i.e., `pCopula`, should be drawn. If true, the line segments are drawn with `col.4 = "#668b5580"`, `lwd.4 = 5`, and `lty.4 = "82"` which you can modify (via the ... below). Applies only when you do *not* set `panel.3d.wireframe` (via the ...).

... additional arguments passed to the underlying `wireframe()`, such as shade, `drape`, `aspect`, etc., or (if you do not specify `panel.3d.wireframe` differently), to the function `panel.3dwire` from the **lattice** package.

Value

An object of class "trellis" as returned by `wireframe()`.

Methods

Wireframe plots for objects of class `"matrix"` , `"data.frame"`, `"Copula"` or `"mvdc"`.

See Also

The `persp-methods` for drawing perspective plots via base graphics.

The **lattice**-based `contourplot2-methods`.

Examples

```
## For 'matrix' objects
## The Frechet--Hoeffding bounds W and M
n.grid <- 26
u <- seq(0, 1, length.out = n.grid)
grid <- expand.grid("u[1]" = u, "u[2]" = u)
W <- function(u) pmax(0, rowSums(u)-1) # lower bound W
M <- function(u) apply(u, 1, min) # upper bound M
x.W <- cbind(grid, "W(u[1],u[2])" = W(grid)) # evaluate W on 'grid'
x.M <- cbind(grid, "M(u[1],u[2])" = M(grid)) # evaluate M on 'grid'
wireframe2(x.W)
wireframe2(x.W, shade = TRUE) # plot of W
wireframe2(x.M, drape = TRUE) # plot of M

## For 'Copula' objects
cop <- frankCopula(-4)
wireframe2(cop, pCopula) # the copula
wireframe2(cop, pCopula, shade = TRUE) # ditto, "shaded"
wireframe2(cop, pCopula, shade = TRUE, col = "gray60") # ditto, "shaded"+grid
wireframe2(cop, pCopula, drape = TRUE, xlab = quote(x[1])) # adjusting an axis label
wireframe2(cop, dCopula, delta=0.01) # the density
wireframe2(cop, dCopula) # => the density is set to 0 on the margins


## For 'mvdc' objects
mvNN <- mvdc(gumbelCopula(3), c("norm", "norm"),
             list(list(mean = 0, sd = 1), list(mean = 1)))
wireframe2(mvNN, dMvdc, xlim=c(-2, 2), ylim=c(-1, 3))
```

xvCopula *Model (copula) selection based on* k*-fold cross-validation*

Description

Computes the leave-one-out cross-validation criterion (or a k-fold version of it) for the hypothesized parametric copula family using, by default, maximum pseudo-likelihood estimation. The leave-one-out criterion is a crossvalidated log likelihood, denoted $\widehat{xv}_n$, defined in equation (42) of Grønneberg and Hjort (2014).

For $k < n$, n the sample size, the k-fold version uses k randomly chosen (almost) equally sized data blocks instead of n. When n is large, k-fold is considerably faster (if k is "small" compared to n).

Usage

```
xvCopula(copula, x, k = NULL, verbose = interactive(),
         ties.method = eval(formals(rank)$ties.method), ...)
```

Arguments

copula	object of class "copula" representing the hypothesized copula family.
x	a data matrix that will be transformed to pseudo-observations.
k	the number of data blocks; if `k = NULL`, `nrow(x)` blocks are considered (which corresponds to leave-one-out cross-validation).
verbose	a logical indicating if progress of the cross validation should be displayed via `txtProgressBar`.
ties.method	string specifying how ranks should be computed if there are ties in any of the coordinate samples of `x` and fitting is based on maximum pseudo-likelihood; passed to `pobs`.
...	additional arguments passed to `fitCopula()`.

Value

A real number equal to the cross-validation criterion multiplied by the sample size.

Note

Note that k-fold cross-validation with `k < n` shuffles the lines of `x` prior to forming the blocks. The result thus depends on the value of the random seed.

The default estimation method is maximum pseudo-likelihood estimation but this can be changed if necessary along with all the other arguments of `fitCopula()`.

References

Grønneberg, S., and Hjort, N.L. (2014) The copula information criteria. *Scandinavian Journal of Statistics* **41**, 436–459.

See Also

fitCopula() for the underlying estimation procedure and gofCopula() for goodness-of-fit tests.

Examples

```
## A two-dimensional data example ----------------------------------
x <- rCopula(200, claytonCopula(3))


## Model (copula) selection -- takes time: each fits 200 copulas to 199 obs.
xvCopula(gumbelCopula(), x)
xvCopula(frankCopula(), x)
xvCopula(joeCopula(), x)
xvCopula(claytonCopula(), x)
xvCopula(normalCopula(), x)
xvCopula(tCopula(), x)
xvCopula(plackettCopula(), x)


## The same with 10-fold cross-validation
set.seed(1) # k-fold is random (for k < n) !
xvCopula(gumbelCopula(),  x, k=10)
xvCopula(frankCopula(),   x, k=10)
xvCopula(joeCopula(),     x, k=10)
xvCopula(claytonCopula(), x, k=10)
xvCopula(normalCopula(),  x, k=10)
xvCopula(tCopula(),       x, k=10)
xvCopula(plackettCopula(),x, k=10)
```

Index