Package ‘copula’

October 26, 2015

Version 0.999-14
VersionNote LastCran: 0.999-13 on 2015-03-05
Date $Date: 2015-10-26 15:14:37 +0100 (Mon, 26. Oct 2015) $, subversion $Revision: 1198 $ <-- don't forget inst/NEWS.Rd !
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 (>= 3.0.1)
Imports stats, graphics, methods, stats4, Matrix, lattice, colorspace, gsl, ADGofTest, stabledist (>= 0.6-4), mvtnorm, pspline
Suggests MASS, KernSmooth, sfsmisc, scatterplot3d, Rmpfr, bbmle, knitr, parallel, mvnormtest, partitions, polynom, randtoolbox, rugarch, Runuran, tseries, VGAM, VineCopula, zoo
SuggestsNote the last line packages {parallel, ..., zoo} are only used in vignettes and demos.
Enhances norlmix
Description Classes (S4) of commonly used elliptical, Archimedean, extreme value and some more copula families. Methods for density, distribution, random number generation, bivariate dependence measures, perspective and contour plots. Fitting copula models including variance estimates. Independence and serial (univariate and multivariate) independence tests, and other copula related tests. Empirical copula and multivariate CDF. Goodness-of-fit tests for copulas based on multipliers, the parametric bootstrap with several transformation options. Merged former package 'nacopula' for nested Archimedean copulas: Efficient sampling algorithms, various estimators, goodness-of-fit tests and related tools and special functions.
License GPL (>= 3) | file LICENCE
ByteCompile yes
**VignetteBuilder**  knitr

**Collate**  AllClass.R  Classes.R  Copula.R  Auxiliaries.R  aux-acopula.R
   huslerReissCopula.R  huslerReissExpr.R  indepCopula.R

**Encoding**  UTF-8

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

**NeedsCompilation**  yes

**Repository**  CRAN

**Date/Publication**  2015-10-26 17:07:26

---

**R topics documented:**

- copula-package ................................................. 4
- .pairsCond ..................................................... 10
- absdPsiMC ..................................................... 12
- acopula-class ............................................... 14
- acR ............................................................ 16
- allComp ....................................................... 18
- An ............................................................. 18
- archmCopula ................................................... 20
- archmCopula-class ........................................... 23
- assocMeasures ................................................ 24
- Bernoulli ...................................................... 25
- beta.Blomqvist ............................................... 27
- C.n ............................................................. 29
- cCopula ........................................................ 31
- coeffG .......................................................... 33
- contour-methods .............................................. 34
- copFamilies ................................................... 36
- Copula .......................................................... 38
- copula-class .................................................. 40
- dDiag .......................................................... 42
- dnacopula ..................................................... 43
- ellipCopula ................................................... 44
- ellipCopula-class ........................................... 46
R topics documented:

emde ................................................. 48
emle ................................................. 50
enacopula ........................................... 52
estim.misc ......................................... 54
evCopula ............................................ 57
evCopula-class ...................................... 58
evTestA ............................................... 59
evTestC ............................................... 61
evTestK ............................................... 62
exchEVTest ......................................... 63
exchTest .............................................. 65
fgmCopula ............................................ 66
fgmCopula-class ...................................... 67
fitCopula ............................................. 68
fitCopula-class ...................................... 73
fitMvdc ............................................... 74
generator ............................................ 76
getAcop ................................................ 78
ggraph-tools ....................................... 79
gnacopula ............................................ 81
gofCopula ............................................ 83
gofEVCopula ......................................... 87
gofOtherTstat ........................................ 89
gofTstat .............................................. 90
gtrafo .................................................. 92
indepCopula .......................................... 95
indepCopula-class ................................... 95
indepTest ............................................ 96
initOpt ............................................... 99
interval .............................................. 101
interval-class ....................................... 102
K ....................................................... 103
log1mexp ............................................ 106
loss ..................................................... 108
math-fun ............................................. 109
multIndepTest ....................................... 110
multSerialIndepTest .................................. 112
Mvdc .................................................. 113
mvdc-class ........................................... 115
nacFrail.time ....................................... 116
nacopula-class ...................................... 117
nacPairthetas ....................................... 118
nesdepth ............................................. 119
onacopula ........................................... 120
opower ............................................... 122
p2P .................................................... 124
pairsRosenblatt ..................................... 125
persp-methods ....................................... 129
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: 0.999-14
VersionNote: LastCran: 0.999-13 on 2015-03-05
Date: $Date: 2015-10-26 15:14:37 +0100 (Mon, 26. Oct 2015) $, subversion $Revision: 1198 $ «– don't forget inst/NEWS.Rd !
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 (>= 3.0.1)
Imports: stats, graphics, methods, stats4, Matrix, lattice, colorspace, gsl, ADGofTest, stabledist (>= 0.6-4), mvtnorm,
Suggests: MASS, KernSmooth, sfsmisc, scatterplot3d, Rmpfr, bbmle, knitr, parallel, mvnormtest, partitions, polynom, randtoolbox, rugarch, Runuran, tseries, VGAM, VineCopula, zoo
SuggestsNote: the last line packages parallel, ..., zoo are only used in vignettes and demos.
Enhances: nor1mix
Description: Classes (S4) of commonly used elliptical, Archimedean, extreme value and some more copula families. Methods ... Efficient sampling algorithms, various estimators, goodness-of-fit tests and related tools and special functions.
License: GPL (>= 3) | file LICENCE
ByteCompile: yes
VignetteBuilder: knitr
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
An Nonparametric Rank-based Estimators of the Pickands Dependence Function
Bernoulli Compute Bernoulli Numbers
C.n The Empirical Copula
Copula Density, Evaluation, and Random Number Generation for Copula Functions
Eulerian Eulerian and Stirling Numbers of First and Second Kind
K Kendall Distribution Function for Archimedean Copulas
Mvdc Multivariate Distributions Constructed from Copulas
RSpobs Pseudo-Observations of Radial and Uniform Part of Elliptical and Archimedean Copulas
SMI.12 SMI Data - 141 Days in Winter 2011/2012
Sibuya Sibuya Distribution - Sampling and Probabilities
absdPsiMC Absolute Value of Generator Derivatives via Monte Carlo
acopula-class  Class "acopula" of Archimedean Copula Families
acopula-families Specific Archimedean Copula Families ("acopula" Objects)
allComp All Components of a (Inner or Outer) Nested Archimedean Copula
archmCopula Construction of Archimedean Copula Class Object
archmCopula-class Class "archmCopula"
beta Sample and Population Version of Blomqvist's Beta for Archimedean Copulas
cCopula Conditional Copula Function
coeffG Coefficients of Polynomial used for Gumbel Copula
contour-methods Methods for Contour Plots in Package 'copula'
copula-class Mother Classes "Copula" and "copula" of All Copulas in the Package
copula-package Multivariate Dependence Modeling with Copulas
dDiag Density of the Diagonal of (Nested) Archimedean Copulas
dncopula Density Evaluation for (Nested) Archimedean Copulas
ebeta Various Estimators for (Nested) Archimedean Copulas
ellipCopula Construction of Elliptical Copula Class Object
ellipCopula-class Class "ellipCopula"
emde Minimum Distance Estimators for (Nested) Archimedean Copulas
emle Maximum Likelihood Estimators for (Nested) Archimedean Copulas
enacopula Estimation Procedures for (Nested) Archimedean Copulas
evCopula Construction of Extreme-Value Copula Class Objects
evCopula-class Classes Representing Extreme-Value Copulas
evTestA Bivariate Test of Extreme-Value Dependence Based on Pickands' Dependence Function
evTestC Large-sample Test of Multivariate Extreme-Value Dependence
evTestK Bivariate Test of Extreme-Value Dependence Based on Kendall's Process
exchEVTest Test of Exchangeability for Certain Bivariate Copulas
exchTest Test of Exchangeability for a Bivariate Copula
fgmCopula Construction of a fgmCopula Class Object
fgmCopula-class Class "fgmCopula"
fitCopula Estimation of the Parameters in Copula Models
fitCopula-class Classes of Fitted Multivariate Models: Copula, Mvdc
fitMvdc Estimation of Multivariate Models Defined via
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAcop</td>
<td>Get &quot;acopula&quot; Family Object by Name</td>
</tr>
<tr>
<td>gnacopula</td>
<td>Goodness-of-fit Testing for (Nested) Archimedean Copulas</td>
</tr>
<tr>
<td>gofBTstat</td>
<td>Various Goodness-of-fit Test Statistics</td>
</tr>
<tr>
<td>gofCopula</td>
<td>Goodness-of-fit Tests for Copulas</td>
</tr>
<tr>
<td>gofEVCopula</td>
<td>Goodness-of-fit Tests for Bivariate Extreme-Value Copulas</td>
</tr>
<tr>
<td>gofTstat</td>
<td>Goodness-of-fit Test Statistics</td>
</tr>
<tr>
<td>gtrafo</td>
<td>GOF Testing Transformations for Archimedean Copulas</td>
</tr>
<tr>
<td>iPsi</td>
<td>Generator Functions for Archimedean and Extreme-Value Copulas</td>
</tr>
<tr>
<td>indepCopula</td>
<td>Construction of Independence Copula Class Objects</td>
</tr>
<tr>
<td>indepCopula-class</td>
<td>Class &quot;indepCopula&quot;</td>
</tr>
<tr>
<td>indepTest</td>
<td>Test Independence of Continuous Random Variables via Empirical Copula</td>
</tr>
<tr>
<td>initOpt</td>
<td>Initial Interval or Value for Parameter Estimation of Archimedean Copulas</td>
</tr>
<tr>
<td>interval</td>
<td>Construct Simple &quot;interval&quot; Object</td>
</tr>
<tr>
<td>interval-class</td>
<td>Class &quot;interval&quot; of Simple Intervals</td>
</tr>
<tr>
<td>log1mexp</td>
<td>Compute $f(a) = \log(1 + e^{-a})$ Numerically Optimal</td>
</tr>
<tr>
<td>loss</td>
<td>LOSS and ALAE Insurance Data</td>
</tr>
<tr>
<td>multIndepTest</td>
<td>Independence Test Among Continuous Random Vectors Based on the Empirical Copula Process</td>
</tr>
<tr>
<td>multSerialIndepTest</td>
<td>Serial Independence Test for Multivariate Continuous Time Series Based on the Empirical Copula Process</td>
</tr>
<tr>
<td>mvdc-class</td>
<td>Class &quot;mvdc&quot;</td>
</tr>
<tr>
<td>nacFrail.time</td>
<td>Timing for Sampling Frailties of Nested Archimedean Copulas</td>
</tr>
<tr>
<td>nacPairthetas</td>
<td>Pairwise Thetas of Nested Archimedean Copulas</td>
</tr>
<tr>
<td>nacopula-class</td>
<td>Class &quot;nacopula&quot; of Nested Archimedean Copulas</td>
</tr>
<tr>
<td>nesdepth</td>
<td>Nesting Depth of a Nested Archimedean Copula (&quot;nacopula&quot;)</td>
</tr>
<tr>
<td>onacopula</td>
<td>Constructing (Outer) Nested Archimedean Copulas</td>
</tr>
<tr>
<td>opower</td>
<td>Outer Power Transformation of Archimedean Copulas</td>
</tr>
<tr>
<td>p2P</td>
<td>Convert (Rho) Matrices to and From Parameter Vectors</td>
</tr>
<tr>
<td>pacR</td>
<td>Distribution of the Radial Part of an Archimedean Copula</td>
</tr>
<tr>
<td>pairsRosenblatt</td>
<td>Plots for Graphical GOF Test via Pairwise Rosenblatt Transforms</td>
</tr>
<tr>
<td>pairwiseCcop</td>
<td>Computations for Graphical GOF Test via Pairwise Rosenblatt Transforms</td>
</tr>
</tbody>
</table>
Methods for Function 'persp' in Package 'copula'

Construction of a Plackett Copula Class Object
Evaluation of (Nested) Archimedean Copulas
Pseudo-Observations
Polylogarithm $\text{Li}_s(z)$ and Debye Functions
Evaluate Polynomials
Print Compact Overview of a Nested Archimedean Copula ("nacopula")
Computing Probabilities of Hypercubes
Q-Q Plot with Rugs and Pointwise Asymptotic Confidence Intervals
Sample Univariate Distributions Involved in Nested Frank and Joe Copulas
Sampling Distribution $F$ for Frank and Joe
Daily Returns of Three Stocks in the Dow Jones
Sampling Exponentially Tilted Stable Distributions
Sampling Logarithmic Distributions
Random nacopula Model
Sampling Nested Archimedean Copulas
Sampling Child 'nacopula's
Random numbers from (Skew) Stable Distributions
"Safety" for Convenience
Serial Independence Test for Continuous Time Series Based on the Empirical Copula Process
Specify the Parameter(s) of a Copula
Methods for 'show' in Package 'copula'
Scatterplot Matrix (splom) with Nice Variable Names
Dependence Measures for Bivariate Copulas
Ali-Mikhail-Haq ("AMH")'s and Joe's Kendall's Tau
Uranium Exploration Dataset of Cook & Johnson (1986)

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)
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.pdf`)

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

**Author(s)**

Marius Hofert, Ivan Kojadinovic, Martin Maechler, and Jun Yan.
Maintainer: Currently, Martin Maechler <maechler@stat.math.ethz.ch>.

References


See Also

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

Examples

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

example(fitCopula)## fitting Copulas
copula

example(fitMvdc)## fitting multivariate distributions via Copulas
copula

example(nacopula)## nested Archimedean Copulas

## Independence Tests: These also draw a ‘Dependogram’:
copula

example(indepTest)## Testing for Independence
copula

example(serialIndepTest)## Testing for Serial Independence
```

Description

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

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

Usage

```r
.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)
See Also

pairsRosenblatt(), the principal function, calling .pairsCond().

---

**absdPsimc**

*Absolute Value of Generator Derivatives via Monte Carlo*

**Description**

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

**Usage**

```r
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 \(\psi(\text{Inf}, *) = 0\).
Details

The absolute value of the $d$th derivative of the Laplace-Stieltjes transform $\psi = \mathcal{L}S[F]$ 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 $V_k \sim F$, $k \in \{1, \ldots, N\}$. 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 $V_k \sim F$, $k \in \{1, \ldots, N\}$ 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.

Author(s)

Marius Hofert (and Martin M.)

References


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...
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 $\psi$ (with $\psi(t) = \exp(-t)$ 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 $\psi$ 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 $(-1)^\text{degree} * \text{absdPsi}$). The number $nMC$ denotes the sample size for a Monte Carlo evaluation approach. If $nMC$ 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.
- **absdPpsi**: 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 $nMC$ and `log` are as described before.
- **score**: A function computing the derivative of the density with respect to the parameter $\theta$.
- **uscore**: A function computing the derivative of the density with respect to 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 \( \psi \) 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 \( \psi \) 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 \( \psi_{01} \) (see dV01) and parameters \( \theta_0 = \text{theta0}, \theta_1 = \text{theta1} \).

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

\[
\psi_{01}(t; V_0) = \exp\left(-V_0\psi_{0}^{-1}(\psi_1(t))\right),
\]

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 \( \psi \) as a function of theta, respectively, theta as a function of Kendall’s tau.

**lambdaL, lambdaU, lambdaLI, lambdaUI:** Compute the lower (upper) tail-dependence coefficient of the bivariate Archimedean copula with generator \( \psi \) 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.

**Author(s)**

Martin Maechler, Marius Hofert

**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.

Examples

```r
## acopula class information
displayClass("acopula")

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

## What are admissible parameters for Clayton copulas?
copClayton@parInterval

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

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

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**

```r
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 $P[X \leq x]$ 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 $p$.

Author(s)

Marius Hofert

References


Examples

```r
## 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 = expression(italic(x)),
  ylab = substitute(italic(f[R](x))~~"for d=1:"~dn, list(dm=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 \( 1: \text{dim}(x) \), whereas its “inner” component copulas will each contain a subset of those indices only.

Usage

allComp(\( x \))

Arguments

\( x \)

an \( \mathbb{R} \) object inheriting from class nacopula.

Value

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

Author(s)

Martin Maechler

Examples

\[
C3 \leftarrow \text{onacopula("AMH", C(0.7135, 1, C(0.943, 2:3)))}
\]
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)
An(x, w)
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.

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 returns 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


See Also


Examples

```r
## 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)
```
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,
ArchmCopula

```r
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 \( \theta \), `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 give Archimedean families, see `copClayton`.

For \( d = 2 \), i.e. \( \text{dim} = 2 \), the AMH, Clayton and Frank copulas allow to model negative Kendall’s \( \tau \) behavior via negative \( \theta \), for AMH and Clayton \(-1 \leq \theta \), and for Frank \(-\infty < \theta \).

For the Ali-Mikhail-Haq copula family ("amhCopula"), only the bivariate case is available.

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

For \( d > 2 \), i.e. \( \text{dim} > 2 \), 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**

See Also

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

`ellipCopula`, `evCopula`.

Examples

```r
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)
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
gumbel.cop <- gumbelCopula(2, dim = 7)
dCopula(x, gumbel.cop)
```
## ArchmCopula-class

Archimedean copula class.

### Description

Objects from the Class

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

| 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, ...)
tailIndex(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


See Also

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

Examples

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

stopifnot(all.equal(tau(gumbel.cop), copGumbel@tau(3)),
           all.equal(tailIndex(gumbel.cop),
                      c(copGumbel@lambdaL(3), copGumbel@lambdaU(3)),
                      check.attributes=FALSE),
           all.equal(iTau (gumbel.cop, 0.681),
                      copGumbel@iTau(0.681)))
```

```r
## 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 \(n\)th Bernoulli number, or generate all Bernoulli numbers up to the \(n\)th, 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 numbers 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)

Author(s)

Martin Maechler

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
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.

Author(s)

Marius Hofert
References

Schmid and Schmidt (2007), Nonparametric inference on multivariate versions of Blomqvist’s beta

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, 0.999999,
        main = "Blomqvist's beta(.) for AMH",
        xlab = expression(theta), ylab = expression(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 = expression(theta), ylab = expression(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 = expression(theta), ylab = expression(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 = expression(theta), ylab = expression(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 = expression(theta), ylab = expression(beta(theta, Gumbel)),
        add=(i > 1), lwd=2, col=cols[i])
mtext("NB: d=2 and d=3 are the same")
legend("bottomright", paste("d =",d.set), bty="n", lwd=2, col=cols)
```

The Empirical Copula

Description

Given a random sample from a distribution with continuous margins and copula C, the empirical copula is a natural nonparametric estimator of C. The function `C.n()` computes the empirical copula.

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

Usage

```r
C.n(u, U, offset=0, method=c("C", "R"))
F.n(x, X, offset=0, method=c("C", "R"))
dCn(u, U, j.ind=1:d, b=1/sqrt(nrow(U)), ...)
```

Arguments

- `u, x, w` an \((m, d)\)-matrix with elements in \([0, 1]\) whose rows contain the evaluation points of the empirical copula.
- `U, X` \((n, d)\)-matrix, for `C.n()` and `Cn()` with elements in \([0, 1]\) and with the same number \(d\) of columns as `u` (or `x` respectively). The rows of `U` are the (pseudo-)data on which the empirical copula is built. The rows of `X` are the “raw” observations. A multivariate random sample (such as `X`) can be transformed to an appropriate `U` via `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 \(\text{sum}(\ldots)/\text{(n+offset)}\); defaults to zero.
- `method` character string indicating which method is applied to compute the empirical CDF or copula. `method="C"` uses a an implementation in C, `method="R"` uses an R implementation.
- `...` additional arguments passed to `C.n()`.
Details

There are several asymptotically equivalent definitions of the empirical copula. Here, the empirical copula is simply defined as the empirical distribution function computed from the pseudo-observations, that is,

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

where $\hat{U}_i, i \in \{1, \ldots, n\}$, denote the pseudo-observations (rows in $U$) and $n$ the sample size.

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

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

where $b$ denotes the bandwidth and $C_n$ the empirical copula.

Value

C.n() and F.n() a numeric vector of length $m$ with the values for C.n() of the empirical copula of $U$ at $u$, and for F.n() of the empirical CDF (cumulative distribution function) of $X$ at $x$.

dCn() returns a $(m,l)$-matrix or an $m$-vector (for $l = 1$; here, $l$ is the length of j.ind), containing the approximated first-order partial derivatives of the unknown copula at $u$.

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 the arguments, and possibly use pobs, i.e., **instead of** Cn(x, u), **use** C.n(u, U=pobs(x)) !

Author(s)

Ivan Kojadinovic, Marius Hofert

References


See Also

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

n <- 100
d <- 3
family <- "Gumbel"
theta <- 2
cop <- onacopulaL(family, list(theta=theta, 1:d))
set.seed(1)
U <- rCopula(n, cop)

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

## compare with true distribution function
mean(abs(pCopula(u, copula=cop)-ec)) # increase n to decrease this error

## compare the empirical copula and the true copula
## on the diagonal of the unit square
Cn. <- function(x) C.n(do.call(cbind, rep(list(x), d)), U=U)
curve(Cn., 0, 1, main=paste("Diagonal of a", family, "copula"),
    xlab="u", ylab=paste(italic(C)[n](italic(u),...,italic(u))))
pC <- function(x) pCopula(do.call(cbind, rep(list(x), d)), cop)
curve(pC, lty=2, add=TRUE)
legend("topleft", lty=1:2, bty="n", inset=0.02,
    legend=c(paste(italic(C)[n]), paste(italic(C))))

## check the empirical copula with its Kendall distribution function
plot( pk(C.n(U, U=U), cop=cop@copula, d=d) ) # must be uniform

## approximate partial derivatives w.r.t. the 2nd and 3rd component
j.ind <- 2:3
der23 <- dCn(u, U=pobs(U), j.ind=j.ind)
der23. <- copula:::dCdU(archmCopula(family, param=theta, dim=d), u=u)[[,j.ind]
summary(as.vector(abs(der23-der23.))) # approximation error summary

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

cCopula

Conditional Copula Function

Description

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

Usage

cCopula(u, copula, j.ind=ncol(u), n.MC=0, log=FALSE)
## Deprecated, for Archimedean copulas only:
cacopula(u, cop, n.MC=0, log=FALSE)

### Arguments

- **u**: \(n \times d\)-matrix; the conditioning is done on the values in the first \(d - 1\) columns.
- **cop**: any "copula" with specified parameters; currently only Archimedean and elliptical copulas provided.
- **j.ind**: indices \(j \geq 2\) for which \(C(u_j \mid u_1, \ldots, u_{j-1})\) is computed.
- **cop**: "outer_nacopula" with specified parameters (only Archimedean copulas are currently provided).
- **n.MC**: Monte Carlo sample size; for Archimedean copulas only, used if positive.
- **log**: if TRUE the logarithm of the conditional copula function is returned.

### Details

cCopula() can be seen as a special case of the Rosenblatt transform which by default computes all conditional distributions. Consequently, cCopula() simply calls rtrafo().

Note that sampling aka random number generation from the conditional distribution can be achieved by rtrafo(*, inverse=TRUE).

### Value

- **numeric** vector of length \(n\) containing the conditional copula function of \(u_d\) given \(u_1, \ldots, u_{d-1}\).

### Note

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

### Author(s)

Marius Hofert, Yongsheng Wang, and Martin Maechler

### See Also

rtrafo; acopula-families.

### Examples

```r
tau <- 0.5
(theta <- copGumbel@iTau(tau)) # 2
d <- 2
# two ways to specify a Gumbel copula:
cop <- onacopulaL("Gumbel", list(theta,1:d))
gCop <- gumbelCopula(theta, dim=d) #
n <- 1000
set.seed(1)
```
coeffG

coeffG <- rCopula(n, cop)

U <- cbind(U[,1], cCopula(U, cop=cop)) # should be ~ U[0,1]^2
plot(U[,1], U[,2])

stopifnot(all.equal(cacopula(U, cop), # with deprecation warning
                   cCopula(U, cop)))

## more examples: --> ?rtrafo

---

**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**

```r
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), \( d \geq 1 \).
- **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.<F00>": uses `dsumSibuya(..., method = "<F00>")`.
- **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

\[
a_k(\theta, d) = (-1)^{d-k} \sum_{j=k}^{d} \alpha_j \ast s(d,j) \ast S(j,k), k \in \{1, \ldots, d\}.
\]
Note

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

As a consequence, the methods and its defaults may change in the future, and so the exact implementation of \texttt{coeffG()} is still considered somewhat experimental.

Author(s)

Marius Hofert and Martin Maechler

Examples

```r
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)
```

Description

Methods for function \texttt{contour} to draw contour lines aka a level plot for objects from package \texttt{copula}.

Usage

```r
## S4 method for signature 'copula'
contour(x, fun,
      n = 51, delta = 0, box01 = TRUE, ...)

## S4 method for signature 'mvdc'
contour(x, fun,
      xlim, ylim, nx = 51, ny = 51,
      xis = seq(xlim[1], xlim[2], length = nx),
      yis = seq(ylim[1], ylim[2], length = ny),
      box01 = FALSE, ...)
```

Arguments

- \(x\): either a "\texttt{copula}" or a "\texttt{mvdc}" object.
- \(\text{fun}\): the \texttt{function} to be plotted; typically \texttt{dCopula} or \texttt{pCopula}.
- \(n\): (for "\texttt{copula}":) the number of points in both directions to do the plotting. The function \texttt{fun} will be evaluated on a grid of size \(n \times n\).
delta a very small number in \([0, \frac{1}{2}]\), defaulting to zero. The x- and y-ranges to be used for plotting will be \([0+\text{delta}, 1-\text{delta}]\), i.e., \([0, 1]\) by default.

xlim, ylim ("mvdc"): the range of the x or y variable, respectively.

nx, ny ("mvdc"): the number of points in x- or y-direction, respectively. The function fun will be evaluated on a grid of size \(nx \times ny\).

xis, yis ("mvdc"): instead of specifying xlim, ylim and nx, ny, the numeric vectors (of length nx and ny) may be specified directly.

box01 logical specifying if faint rectangle should be drawn at the \([0, 1]^2\) borders (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

```r
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))
```
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 specifically, 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

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

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

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

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

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

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

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

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

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

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 \( \theta_0, \theta_1 \) satisfy \( \theta_0 \leq \theta_1 \).

Value

A "acopula" object.

Author(s)

Marius Hofert

References


See Also

The class definition, "acopula".
getAcop accesses these families "programmatically".

Examples

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

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

copAMH

## 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="^copula[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 = expression(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
  lu <- pmin(10, I[2])
  curve(Cf@tau(x), I[1], lu, col = 3,
        xlab = bquote(theta ~ Vformat(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 <- rncopula(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(\cdot)) number of observations to be generated.

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


See Also

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

Examples

```r
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
c13 <- claytonCopula(2, dim = 3)
v <- rCopula(1000, c13)
pairs(v)
if(require(scatterplot3d))
  scatterplot3d(v)

## Compare with the "nacopula" version :
fu1 <- dCopula(v, c13)
fu2 <- copClayton@dCopula(v, theta = 2)
Fu1 <- pCopula(v, c13)
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" and "copula" of All Copulas in the Package

Description

A copula is a multivariate distribution with uniform margins. The virtual class "Copula" is the mother of all copula classes in the package copula which encompasses classes of the former packages nacopula and copula.

The virtual class "copula" is the mother of all copula classes from former package copula.
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.lowbd`: lower bounds for the parameters, of class "numeric".
- `param.upbd`: upper bounds for the parameters, of class "numeric".
- `fullname`: 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

```r
hc <- evCopula("husler", 1.25)
dim(hc)
smoothScatter(u <- rCopula(2*11, hc))
tailIndex(hc)
tau(hc)
rho(hc)
str(hc)
```
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 $u$.

Author(s)
Martin Maechler, Marius Hofert

References

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,
Density Evaluation for (Nested) Archimedean Copulas

Description

For a (nested) Archimedean copula (object of class `nacopula`) \( x \), \( \text{dcopula}(u, x) \) (or also currently still \( \text{dnacopula}(x, u) \)) evaluates the density of \( x \) at the given vector or matrix \( u \).

Usage

```r
## S4 method for signature 'matrix,nacopula'
\text{dCopula}(u, \text{copula, log=FALSE, } ...)

## *Deprecated*:
\text{dnacopula}(x, u, \text{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 \text{log} of the density should be returned.
- `...`: optional arguments passed to the copula’s \text{dacopula} function (slot), such as \text{nMC} (non-negative integer) for possible Monte Carlo evaluation (see \text{dacopula} in \text{acopula}).

Details

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

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

Value

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

Note

\text{dCopula}(u, \text{copula}) is a generic function with methods for all our copula classes, see \text{dCopula}. 
Author(s)
Marius Hofert, Martin Maechler

References


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

Examples
```r
## 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))

dCopula(u, C20)

## Evaluate the copula density at the point u = (0.5,...,0.5)
theta <- rep(.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 Object*

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

Usage

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

Arguments

- **family**: a character string specifying the family of an elliptical copula. Implemented families are "normal" and "t".
- **param**: a numeric vector specifying the parameter values. The `getRho()` method accesses this vector, whereas `p2P()` and `getSigma()` provide the corresponding "Rho" 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. Implemented structures are "ex" for exchangeable, "ar1" for AR(1), "toep" for Toeplitz, and "un" for unstructured.
- **df**: an 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`.

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 1e-01. 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 “Rho” (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)# the P ("Rho") matrix (with diagonal = 1)

## 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"
Description

Copulas generated from elliptical multivariate distributions.

Objects from the Class

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

Slots

dispstr: Object of class "character", indicating how the dispersion matrix is parameterized; can 'ex', 'ar1', 'toep', or 'un'.
dimension: Object of class "numeric", dimension of the copula.
parameters: Object of class "numeric", parameter value.
param.names: Object of class "character", parameter names.
param.lower: Object of class "numeric", parameter lower bound.
param.upper: Object of class "numeric", parameter upper bound.
fullname: Object of class "character", family names of the copula.

Extends

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

Methods

dCopula signature(copula = "normalCopula"): ...
pCopula signature(copula = "normalCopula"): ...
rCopula signature(copula = "normalCopula"): ...
dCopula signature(copula = "tcopula"): ...
pCopula signature(copula = "tcopula"): ...
rCopula signature(copula = "tcopula"): ...

See Also

ellipCopula, copula-class.
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 × 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, \textit{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 (\texttt{include.K=TRUE}) or not. Second, using either the sum of squares of the standard normal quantile function (\texttt{method="mde.chisq.CvM"} and \texttt{method="mde.chisq.KS"}) or the sum of negative logarithms (\texttt{method="mde.gamma.CvM"} and \texttt{method="mde.gamma.KS"}), a map to a chi-square or an Erlang distribution is applied, respectively. Finally, a Cramér-von Mises (\texttt{method="mde.chisq.CvM"} and \texttt{method="mde.gamma.CvM"}) or Kolmogorov-Smirnov (\texttt{method="mde.chisq.KS"} and \texttt{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

\texttt{list} as returned by \texttt{optimize}, including the minimum distance estimator.

Author(s)

Marius Hofert

References


See Also

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

Examples

\begin{verbatim}
tau <- 0.25
(theta <- copGumbel@iTau(tau)) # 4/3
d <- 20
(cop <- onacopula(“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)
}\end{verbatim}
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 \(d\)th 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 "optimize", 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).

Author(s)

Martin Maechler, Marius Hofert.

References


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)
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(), ...)
enacopula

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 \( n \cdot MC \) 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.
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’s beta.
- **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

\[ \text{ebeta}(u, \cop, \interval = \text{initOpt}(\cop@\copula@\text{name}), \ldots) \]
\[ \text{edmle}(u, \cop, \interval = \text{initOpt}(\cop@\copula@\text{name}), \text{warn=}\text{TRUE}, \ldots) \]
\[ \text{etau}(u, \cop, \text{method} = c("\tau\text{.mean", "\theta\text{.mean"}), \text{warn=}\text{TRUE}, \ldots) \]

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.

\( \cop \)  \text{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).

\( \text{method} \)  a character string specifying the method (only for \( \eta\text{tau} \)), which has to be one (or a unique abbreviation) of

- "\tau\text{.mean}" method-of-moments-like estimator based on the average of pairwise sample versions of Kendall’s tau;
- "\theta\text{.mean}" average of the method-of-moments-like Kendall’s tau estimators.

\( \text{warn} \)  logical indicating if warnings are printed:

- \text{edmle()} for the family of "Gumbel" if the diagonal maximum-likelihood estimator is smaller than 1.
- \text{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.

\( \ldots \)  additional arguments passed to \text{cor} (for \( \eta\text{tau} \)), to \text{optimize} (for \text{edmle}), or to \text{safeUroot} (for \text{ebeta}).

Details

For \text{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 \text{edmle}, the parameter is estimated via maximum-likelihood estimation based on the diagonal.

For \text{etau}, the \text{method}="\tau\text{.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 \text{method}="\theta\text{.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, \text{etau} should be used with the (default) method "\tau\text{.mean}" since "\theta\text{.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.

Author(s)

Marius Hofert

References


See Also

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

Examples

```r
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)

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.
```
evCopula

Construction of Extreme-Value Copula Class Objects

Description

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

Usage

```r
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.

See Also

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

```r
## Notice that, for a given degree of dependence,
## these copulas are strikingly similar.

tau <- 0.33

gumbel.cop <- evCopula("gumbel")
stopifnot(identical(gumbel.cop, gumbelCopula(0.33))
gumbel.cop@parameters <- iTau(gumbel.cop, tau)

galambos.cop <- galambosCopula()
galambos.cop@parameters <- iTau(galambos.cop, tau)

huslerReiss.cop <- huslerReissCopula()
huslerReiss.cop@parameters <- iTau(huslerReiss.cop, tau)

tawn.cop <- tawnCopula()
tawn.cop@parameters <- iTau(tawn.cop, tau)

tev.cop <- tevCopula()
tev.cop@parameters[1] <- iTau(tev.cop, tau)

curve(A(gumbel.cop, x, 0, 1,
       main = "A(x) for five Extreme Value cop. w/ tau = 1/3")
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, add=TRUE)

## the t-EV-copula has always positive tau :
curve(vapply(x, function(x) tau(tevCopula(x)), 0.), -1, 1,
       col=2, n=257, ylim=0:1, ylab=quote(tau), xlab=quote(rho),
       main= expression(tau( tevCopula(rho ))))
```

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"))

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.
Details

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

Value

Returns a list whose attributes 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. One way of dealing with ties was suggested in the last reference.

References


See Also


Examples

```r
## 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")
```

```r
evTestA(uC)
```
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

Returns a list whose attributes 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. One way of dealing with ties was suggested in the last reference.

References


See Also

Examples

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

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

---

**evTestK**  
*Bivariate Test of Extreme-Value Dependence Based on Kendall’s Process*

**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**

```r
evTestK(x, method = c("fsample","asymptotic","jackknife"))
```

**Arguments**

- `x`: a data matrix.
- `method`: specifies the variance estimation method; can be either "fsample" (finite-sample, the default), "asymptotic" or "jackknife".

**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**

Returns a list whose attributes are:

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

**References**


**See Also**

Examples

```r
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"))
## Not run:
(K3 <- evTestK(Uc, method = "jackknife") )# slow !
## End(Not run)
```

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.

Usage

```r
exchEVTest(x, N = 1000, estimator = "CFG", derivatives = "Cn", m = 100)
```

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.
- `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).
- `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 first two references.
Value

Returns a list whose attributes are:

- **statistic**: value of the test statistic.
- **pvalue**: 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. One way of dealing with ties was suggested in the last reference.

References


See Also

- `exchTest`, `gofCopula`.

Examples

```r
## Do these data come from exchangeable copulas?
exchEVTest(rCopula(200, gumbelCopula(3)))
exchEVTest(rCopula(200, claytonCopula(3)))

## Creating asymmetric data
rKhoudraji <- function(cop, n, a=0.6, b=0.95)
{
  u <- rCopula(n, cop)
  v <- matrix(runif(2*n),n,2)
  cbind(pmax(u[,1]^(1/a),v[,1]^(1/(1-a))),
       pmax(u[,2]^(1/b),v[,2]^(1/(1-b))))
}
exchEVTest(rKhoudraji( gumbelCopula(3),200))
exchEVTest(rKhoudraji(claytonCopula(3),200))
```
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.

Usage

```r
exchTest(x, N = 1000, m = 0)
```

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.
- `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 two first references.

Value

Returns a list whose attributes 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. One way of dealing with ties was suggested in the last reference.

References

fgmCopula

See Also

exchEVTest, gofCopula.

Examples

# Do these data come from exchangeable copulas?
exchTest(rCopula(200, gumbelCopula(3)))
exchTest(rCopula(200, claytonCopula(3)))

# Creating asymmetric data
khoudraji <- function(cop, n, a = 0.6, b = 0.95)
  {
    u <- rCopula(n, cop)
    v <- matrix(runif(2*n), n, 2)
    x <- cbind(pmax(u[,1]^(1/a), v[,1]^(1/(1-a))),
              pmax(u[,2]^(1/b), v[,2]^(1/(1-b))))
    x
  }
exchTest(khoudraji(gumbelCopula(3), 200))
exchTest(khoudraji(claytonCopula(3), 200))

fgmCopula

Construction of a fgmCopula Class Object

Description

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

Usage

fgmCopula(param, dim = 2)

Arguments

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

See Also
`Copula`, `copula-class`, `fitCopula`.

Examples
```r
## 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), dim = 3)

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

Description
Multivariate Multiparameter Farlie-Gumbel-Morgenstern Copula.

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

Slots
- `exprdist`: Object of class "expression", expressions for the cdf and pdf of the copula. These expressions are used in function `pcopula()` and `dcopula()`.
- `dimension`: Object of class "numeric", the dimension of the copula.
- `parameters`: Object of class "numeric", parameter values.
- `param.names`: Object of class "character", parameter names.
- `param.lowbnd`: Object of class "numeric", parameter lower bound.
- `param.upbnd`: Object of class "numeric", parameter upper bound.
- `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


See Also

copula-class, fgmCopula-class.

Description

Estimation of the Parameters in Copula Models

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

Usage

loglikCopula(param, x, copula, hideWarnings)
fitCopula(copula, data, method = c("mpl", "ml", "itau", "irho"),
 start = NULL, lower = NULL, upper = NULL,
 optim.method = "BFGS", optim.control = list(maxit=1000),
 estimate.variance = NA, hideWarnings = TRUE)
Arguments

param a vector of parameter values.

x, data \( n \times d \)-matrix of (pseudo-)observations (for "mpl" and "ml" with values necessarily 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 values in \([0, 1]\).

copula a "copula" object.

hidewarnings `deprecated` and unused for `loglikCopula()`; logical; if TRUE, warning messages from likelihood maximization (mostly evaluating at invalid parameter values) are suppressed.

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

start a vector of starting values for `param`.

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

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

optim.method the method for `optim()`.

estimate.variance logical; if true (as by default, if the optimization converges), the asymptotic variance is estimated.

Details

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

Value

`loglikCopula()` returns the log likelihood evaluated at the given value of "param".

The return value of `fitCopula()` is an object of class "fitCopula" (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`).

- `copula` the fitted copula.

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

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

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

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

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()).

For the t copula with df.fixed=FALSE (see ellipCopula()), the methods "itau" and "irho" cannot be used in fitCopula(). For the methods "ml" and "mpl1", when start is not specified, the starting value for df is set to copula@df, typically 4. Also, the asymptotic variance cannot (yet) be estimated for method "mpl1".

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

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

References


See Also

Copula, mvdc for fitting multivariate distributions including the margins aka “meta copula”s; gofCopula.

For maximum likelihood of (nested) archimedean copulas: emle, etc.

Examples

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

(Xtras <- copula:::doExtras())
n <- if(Xtras) 200 else 64

set.seed(7) # for reproducibility
x <- rCopula(n, gumbel.cop)# "true" observations
u <- pobs(x) # pseudo-observations
fit.tau <- fitCopula(gumbel.cop, u, method="itau")
fit.tau
coef(fit.tau)# named vector
fit.rho <- fitCopula(gumbel.cop, u, method="irho")
fit.rho
fit.mpl <- fitCopula(gumbel.cop, u, method="mpl")
fit.mpl
fit.ml <- fitCopula(gumbel.cop, x, method="ml")
fit.ml # print()ing works via summary() ...

# and of that, what's the log likelihood (in two different ways):
ll <- logLik(fit.ml)
stopifnot(all.equal(as.numeric(ll),
    loglikCopula(coef(fit.ml), x=x, copula=gumbel.cop)))

# a multiparameter example
set.seed(6)
normal.cop <- normalCopula(c(0.6,0.36, 0.6),dim=3, dispstr="un")
x <- rCopula(n, normal.cop)# "true" observations
u <- pobs(x) # pseudo-observations
fit.tau <- fitCopula(normal.cop, u, method="itau")
fit.tau
fit.rho <- fitCopula(normal.cop, u, method="irho")
fit.rho
fit.mpl <- fitCopula(normal.cop, u, method="mpl")
fit.mpl
coef(fit.mpl) # named vector
str(sf.mpl <- summary(fit.mpl))
coef(sf.mpl)# the matrix, with SE, t-value, ...

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

## a t copula with 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
u <- pobs(x) ## pseudo-observations
## maximum likelihood; start := (rho[1:3], df)
(tc.ml <- fitCopula(tCop, x, method="ml", start=c(0,0,0,10)))
(tc.ml. <- fitCopula(tCop, x, method="ml")) # without 'start'
## maximum pseudo-likelihood; the asymptotic variance cannot be estimated
(tc.mpl <- fitCopula(tCop, u, method="mpl", estimate.variance=FALSE,
                      start= c(0,0,0,10)))

if(Xtras) { ##---- typically not run with CRAN checking: ---
## without start:
(tc.mp. <- fitCopula(tCop, u, method="mpl", estimate.variance=FALSE))
all.eqCop <- function(x,y, ...) {
    x@fitting.stats$count <- y@fitting.stats$count <- NULL
    all.equal(x,y, ...) }
stopifnot(all.eqCop(tc.ml, tc.ml, tolerance=.005),
          all.eqCop(tc.mpl, tc.mp, tolerance=.005))

## same t copula but with df.fixed=TRUE (--> use same data!)
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.
- `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".

```r
(tc.f <- tCopula(c(0.2, 0.4, 0.6), dim=3, dispstr="un", df=5, df.fixed=TRUE))
## maximum likelihood; start := rho[1:3]           
(tc.ml <- fitCopula(tc.f, x, method="ml", start=c(0,0,0)))
(tc.ml <- fitCopula(tc.f, x, method="ml"))## without 'start'
stopifnot(all.eqCop(tc.ml,tc.ml., tolerance= 4e-4))
## the (estimated, asymptotic) var-cov matrix:
vcov(tc.ml)

## maximum pseudo-likelihood; the asymptotic variance cannot be estimated
(tc.mp <- fitCopula(tc.f, x, method="mpl", estimate.variance=FALSE, 
                    start=c(0,0,0)))
(tc.mp. <- fitCopula(tc.f, x, method="mpl", estimate.variance=FALSE))
stopifnot(all.eqCop(tc.mp,tc.mp., tolerance= 1e-5))

}## end{typically not run ...}
```
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


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, hideWarnings)
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, ...)
## 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.
fitMvdc

mvdc a "mvdc" object.
hideWarnings deprecated and unused for loglikMvdc; logical; if TRUE, warning messages from likelihood maximization (mostly evaluating at invalid parameter values) are suppressed.
data a data matrix.
start a vector of starting value for "param". See "param" above for ordering of this vector.
optim.control a list of controls to be passed to optim.
method the method for optim.
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.
object an R object of class "fitMvdc".
... 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 demo(QARClayton) prepared by Roger Koenker <rkoenker@uiuc.edu>.

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

Finally, note that 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).
See Also

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

For fitting univariate marginals, fitdistr().

Examples

gumbel.cop <- gumbelCopula(3, dim=2)
gMvd2 <- mvdc(gumbel.cop, c("exp","exp"),
          list(list(rate=2), list(rate=4)))
set.seed(1)
x <- rMvd(10000, gMvd2)
## with identical margins:
gMvd.I <- mvdc(gumbel.cop, "exp", param = list(rate=3), marginsIdentical=TRUE)
if(copula:::doExtras()) { ## these are typically not run with CRAN checking:
## Takes about 25 sec. [2012-07]:
  fit2 <- fitMvd(x, gMvd2, start = c(1,1, 2),
             hideWarnings=FALSE) ## <- show warnings here
  print(fit2)
  ## The (estimated, asymptotic) var-cov matrix:
  print( vcov(fit2) )

  fitI <- fitMvd(x, gMvd.I, start = c(3, 2),
             optim.control=list(trace= TRUE, REPORT= 2))
  print(coef(summary(fitI)))
  print(fitI)

## a wrong starting value can already be *the* problem:
  f2 <- try(fitMvd(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:
  fit.I.2 <- fitMvd(x, gMvd.I, start = c(1, 1), method = "Nelder",
             optim.control=list(trace= TRUE))

## The (estimated, asymptotic) var-cov matrix:
  print( vcov(fit2) )
  str(sfi <- summary(fitI))
  stopifnot(is.matrix(coef(sfi)))
}

## Roger Koenker prepared a demo illustrating MLE for a Clayton AR(1)
## >>>> see vignette("AR_Clayton", package="copula") % ../vignettes/AR_Clayton.Rmd
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

\[
\begin{align*}
\text{psi} & (\text{copula}, s) \\
\text{iPsi} & (\text{copula}, u, \ldots) \\
\text{diPsi} & (\text{copula}, u, \text{degree}=1, \log=\text{FALSE}, \ldots) \\
\text{A} & (\text{copula}, w) \\
\text{dAdu} & (\text{copula}, w)
\end{align*}
\]

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 \texttt{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

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

where \( A : [0, 1] \to [1/2, 1] \) is convex and satisfies \( \max(t, 1 - t) \leq A(t) \leq 1 \) 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


generator 77
See Also

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

Examples

```r
## 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

```r
getAcop(family, check=TRUE)
```

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".

Value

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

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

Author(s)

Martin Maechler
See Also

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

Examples

```r
ggetAcop("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))

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

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

```r
pairwiseCcop(u, cop, ...)
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.

**cop**
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[1,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

Author(s)

Marius Hofert, Martin Mächler.

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 rtrafo() 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().

  "Hering.Hofert" for the multivariate transformation of Rosenblatt (1952); see rtrafo.
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 `rtrafo()` 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 `estimNmethod` 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.

Author(s)

Marius Hofert, Martin Maechler.
References


See Also
gofTstat() for the implemented test statistics, gtrafo() for the multivariate transformation(s) htrafo() and rtrafo() 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

Goodness-of-fit tests for copulas based on the empirical process comparing the empirical copula with a parametric estimate of the copula derived under the null hypothesis. Approximate p-values for the test statistic can be obtained either using the *parametric bootstrap* (see the two first references) or by means of a fast *multiplier* approach (see references three and four).

The default test statistic, "Sn", is the Cramer-von Mises functional $S_n$ defined in Equation (2) of Genest, Remillard and Beaudoin (2009).

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

Usage
gofCopula(copula, x, N = 1000,
method = "Sn",
estim.method = eval(formals(fitCopula)$method),
simulation = c("pb", "mult"),
verbose = TRUE, print.every = NULL, ...)

gofPB(copula, x, N, method = eval(formals(gofTstat)$method),
estim.method = eval(formals(fitCopula)$method),
trafo.method = c("none", "rtrafo", "htrafo"),
trafoArgs = list(), verbose = TRUE, ...)

gofMB(copula, x, N, method = c("Sn", "Rn"),
estim.method = eval(formals(fitCopula)$method), verbose = TRUE,
useR = FALSE, m = 1/2, zeta.m = 0, b = 0.05, ...)
Arguments

copula  object of class "copula" representing the hypothesized copula family.
x       a data matrix that will be transformed to pseudo-observations.
N       number of bootstrap or multiplier replications to be used to simulate 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", "SnC", "AnChisq", or "AnGamma", see gofTstat(). 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 simulation method for generating realizations of the test statistic under the null hypothesis; can be either "pb" (parametric bootstrap) or "mult" (multiplier).
print.every  is deprecated in favor of verbose.
verbose  a logical specifying if progress of the 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 optim.method, optim.control, lower, or upper depending on the optim.method.
trafo.method  string specifying the transformation to $U[0,1]^d$; either "none" or one of "rtrafo", see rtrafo, or "htrafo", see htrafo.
trafoArgs  a list of optional arguments passed to the transformation method (see trafo.method above).
useR  logical indicating whether an R or the C implementation is used.
m, zeta.m, b  only for method "Rn" in "MB", the multiplier bootstrap. m is the power, zeta.m the adjustment parameter $\zeta_m$ for the denominator of the test statistic, and 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 either by maximizing the pseudo-likelihood, by inverting Kendall's tau, or by inverting Spearman's rho. If the multiplier is used, any estimation method can be used in the bivariate case, but only maximum pseudo-likelihood estimation can be used in the multivariate (multiparameter) case.

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

Thus far, the multiplier approach is implemented for six copula families: the Clayton, Gumbel, Frank, Plackett, normal and t.
Although the processes involved in the multiplier and the parametric bootstrap-based test are asymptotically equivalent under the null, note that the finite-sample behavior of the two tests might differ significantly.

Also note that in the case of the parametric and multiplier bootstraps, the approximate p-value is computed as

\[
(0.5 + \sum_{b=1}^{N} \mathbb{1}_{\{T_b \geq T\}})/(N + 1),
\]

where \( T \) and \( T_b \) denote the test statistic and the bootstrapped test statistic, 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.

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 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-values. One way of dealing with ties was suggested in the Journal of Statistical Software reference.

Since \( R \) is widely used by practitioners, a word of warning concerning goodness-of-fit tests in general is also advisable. Goodness-of-fit tests are often (ab)used in practice to “justify” an assumption under which one then continues to work (carelessly). From a mathematical point of view, this is not correct.

References


See Also

`fitCopula()` for the underlying estimation procedure and `gofTstat()` for the available test statistics.

Examples

```r
## the following example is available in batch through
## demo(gofCopula) &gt;= &lt;/demo/gofCopula.R &lt;-- keep &gt;&gt; EXACTLY &lt;&lt; in sync!&lt;

## Not run:
## A two-dimensional data example -------------------------------
x &lt;- rCopula(200, claytonCopula(3))

(tau. &lt;- cor(x, method="kendall")[1,2]) # around 0.5 -- 0.6
## Does the Gumbel family seem to be a good choice?
(thG &lt;- iTau(gumbelCopula(), tau.)) # 3.02
gofCopula(gumbelCopula(thG), x)
# SnC: really s..1..o..w.. --- SnB is *EVEN* slower
gofCopula(gumbelCopula(thG), x, method = "SnC")
## What about the Clayton family?
(thC &lt;- iTau(claytonCopula(), tau.)) # 4.05
gofCopula(claytonCopula(thC), x)
gofCopula(claytonCopula(thC), x, method = "AnChisq")

## The same with a different estimation method
gofCopula(gumbelCopula(thG), x, estim.method="itau")
gofCopula(claytonCopula(thC), x, estim.method="itau")

## A three-dimensional example -------------------------------
x &lt;- rCopula(200, tCopula(c(0.5, 0.6, 0.7), dim = 3, dispstr = "un"))

## Does the Clayton family seem to be a good choice?
## here starting with the "same" as indepCopula(3):
(gC13 &lt;- gumbelCopula(1, dim = 3, use.indepC="FALSE"))
gofCopula(gC13, x)
## What about the t copula?
t.copula &lt;- tCopula(rep(0, 3), dim = 3, dispstr = "un", df.fixed=TRUE)
## this is *VERY* slow currently
gofCopula(t.copula, x)

## The same with a different estimation method
gofCopula(gC13, x, estim.method="itau")
gofCopula(t.copula, x, estim.method="itau")

## The same using the multiplier approach
gofCopula(gC13, x, simulation="mult")
gofCopula(t.copula, x, simulation="mult")

## End(Not run)
```
Description

Goodness-of-fit tests for extreme-value copulas based on the empirical process comparing one of the two nonparametric 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 $S_n$ 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 = "mpl",
    estimator = "CFG", m = 1000, verbose = TRUE,
    print.every = NULL, optim.method = "Nelder-Mead")

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), "i.tau" (inversion of Kendall’s tau) or "i.rho" (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.

print.every is deprecated in favor of verbose.

verbose a logical specifying if progress of the bootstrap should be displayed via txtProgressBar.

optim.method the method for "optim".

Details

More details can be found 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.
- **parameter**: estimates of the parameters for the hypothesized copula family.

Note

For a given degree of dependence, the most popular extreme-value copulas are strikingly similar.

References


See Also


Examples

```r
## Not run:
x <- rCopula(100, claytonCopula(3))

## Does the Gumbel family seem to be a good choice?
gofEVCopula(gumbelCopula(1), x)

## The same with different estimation methods
gofEVCopula(gumbelCopula(1), x, method="itau")
gofEVCopula(gumbelCopula(1), x, method="irho")

## The same with different extreme-value copulas
gofEVCopula(galambosCopula(1), x)
gofEVCopula(galambosCopula(1), x, method="itau")
gofEVCopula(galambosCopula(1), x, method="irho")
gofEVCopula(huslerReissCopula(1), x)
gofEVCopula(huslerReissCopula(1), x, method="itau")
gofEVCopula(huslerReissCopula(1), x, method="irho")
gofEVCopula(tevcopula(0, df.fixed=TRUE), x)
gofEVCopula(tevcopula(0, df.fixed=TRUE), x, method="itau")
gofEVCopula(tevcopula(0, df.fixed=TRUE), x, method="irho")

## End(Not run)
```
### 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 \( \mathbb{R}^d \).

### Value

An \((n, d - 1)\)-matrix where the \((i, k)\)th entry is

\[
B_{ik} = \frac{\sum_{j=1}^{k} u_{ij}^2}{\sum_{j=1}^{d} u_{ij}^2}.
\]

### Author(s)

Marius Hofert.

### References


### Examples

```r
## 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, \ldots, d-1\}
Bmat <- gofBTstat(U)

## (graphically) check if Bmat[,k] follows a Beta(k/2, (d-k)/2) distribution
qqplot2(Bmat[,k], qf=function(p) qbeta(p, shape1=k/2, shape2=(ncol(Bmat)+1-k)/2),
       main.args=list(text=as.expression(substitute(paste("Beta",s1,s2)))))
```
gofTstat

**Goodness-of-fit Test Statistics**

**Description**

gofTstat() computes various goodness-of-fit test statistics typically used in gofCopula(*, simulation = "pb").

**Usage**


**Arguments**

- **u** `n × 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 \(i \in \{1,...,n\}\).

- **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 chi-square distribution with \(d\) degrees of freedom. To be more precise, the Anderson-Darling test statistic of the variates

\[
\chi^2_d \left( \sum_{j=1}^{d} (\Phi^{-1}(u_{ij}))^2 \right)
\]

is computed (via ADGofTest::ad.test), where \(\Phi^{-1}\) denotes the quantile function of the standard normal distribution function, \(\chi^2_d\) denotes the distribution function of the chi-square distribution with \(d\) degrees of freedom, and \(u_{ij}\) is the \(j\)th component in the \(i\)th row of \(u\).

- "AnGamma" similar to method="AnChisq" but based on the variates

\[
\Gamma_d \left( \sum_{j=1}^{d} (-\log u_{ij}) \right),
\]

where \(\Gamma_d\) 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 \texttt{R} or \texttt{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 \texttt{numeric}.

Author(s)

Marius Hofert, Martin Maechler.

References


See Also

\texttt{gofCopula()} for goodness-of-fit tests where (some of) these test statistics are used.

Examples

\begin{verbatim}
## 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 <- rtrafo(Uhat, cop) # Rosenblatt transformed data (with correct copula)
gofTstat(u, method="Sn", copula=cop) # compute test statistic Sn; requires copula argument
\end{verbatim}
Description

Compute the following goodness-of-fit (GOF) testing transformations,

**Rosenblatt’s transformation** `rtrafo()` is also of importance outside of GOF computations: The Rosenblatt transformation is used for computing conditional copulas and for sampling purposes. Currently, `rtrafo()` is applicable to elliptical and Archimedean copulas.

`htrafo()` the transformation described in Hering and Hofert (2014), for Archimedean copulas.

Usage

`rtrafo(u, cop, j.ind = NULL, n.MC=0, inverse=FALSE, log=FALSE)`
`htrafo(u, cop, 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 `cop` 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 (`cop`) ones.

- **cop** a "Copula" with specified parameters based on which the transformation is computed. For `htrafo()`, currently only Archimedean copulas are supported (specified as "outer_nacopula"), whereas for `rtrafo()`, hierarchical Archimedean and elliptical copulas (see `ellipCopula`) are allowed.

- **j.ind** `NULL` (in which case the Rosenblatt transformation is computed (all components)) or an `integer` between 2 and `d` indicating the conditional distribution which is to be computed.

- **n.MC** parameter `n.MC` for `K` (for `htrafo`) or for approximating the derivatives involved in the Rosenblatt transform for Archimedean copulas (for `rtrafo`).

- **inverse** logical indicating whether the inverse of the transformation is returned.

- **log** logical specifying if the logarithm of the transformation, i.e., conditional distributions is desired.

- **include.K** logical indicating whether the last component, involving the Kendall distribution function `K`, is used in `htrafo`.

- **method** method to compute `qK`.

- **u.grid** argument of `qK` for `method="discrete"`.

- **...** additional arguments passed to `qK()` if `inverse` is true.
Details

rtrafo Given a $d$-dimensional random vector $U$ following an Archimedean copula $C$ with generator $\psi$, the conditional copula of $U_j = u_j$ given $U_1 = u_1, \ldots, U_{j-1} = u_{j-1}$ is given by

$$C(u_j \mid u_1, \ldots, u_{j-1}) = \frac{\psi^{(j-1)} \left( \sum_{k=1}^{j} \psi^{-1}(u_k) \right)}{\psi^{(j-1)} \left( \sum_{k=1}^{j-1} \psi^{-1}(u_k) \right)}.$$

This formula is either evaluated with the exact derivatives or, if $n\text{MC}$ is positive, via Monte Carlo; see \texttt{absdPsiMC}.

Rosenblatt (1952) showed that $U' \sim U[0,1]^m$, where $U'_1 = U_1, U'_2 = C(U_2 \mid U_1), \ldots$, and $U'_m = C(U_m \mid U_1, \ldots, U_{m-1})$.

rtrafo applies this transformation row-wise to $u$ (with default $m = d$) and thus returns an $n \times m$-matrix.

The inverse transformation (\texttt{inverse=}TRUE) applied to $U'[0,1]^d$ data is known as “conditional distribution method” for sampling.

Note that for the Clayton, the Gauss and the t copula, both the conditional copulas and their inverses are known explicitly and rtrafo() utilizes these explicit forms.

htrafo Given a $d$-dimensional random vector $U$ following an Archimedean copula $C$ with generator $\psi$, Hering and Hofert (2014) showed that $U' \sim U[0,1]^d$, where

$$U'_j = \left( \frac{\sum_{k=1}^{j} \psi^{-1}(U_k)}{\sum_{k=1}^{j+1} \psi^{-1}(U_k)} \right)^j, \; j \in \{1, \ldots, d-1\}, \; U'_d = K(C(U)).$$

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 (\texttt{include.K=}TRUE) or not (\texttt{include.K=}FALSE).

Value

htrafo() returns an $n \times d$- or $n \times (d-1)$-matrix (depending on whether \texttt{include.K} is \texttt{TRUE} or \texttt{FALSE}) containing the transformed input $u$.

rtrafo() returns an $n \times d$-matrix containing the transformed input $u$.

Author(s)

Marius Hofert, Martin Maechler.

References


**See Also**

gofCopula where both transformations are applied or emde where htrafo is applied.

**Examples**

tau <- 0.5
(theta <- copGumbel@iTau(tau)) # 2
(copG <- onacopula("Gumbel", list(theta, 1:5))) # d = 5

set.seed(1)
n <- 1000
x <- rnacopula(n, copG)
x <- qnorm(x) # 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-Gumbel model
## with the transformation of Hering and Hofert (2014):
u.h <- htrafo(u, cop=copG) # transform the data
pairs(u.h, gap=0, cex=0.2) # looks good

## with the transformation of Rosenblatt (1952):
u.r <- rtrafo(u, cop=copG) # transform the data
pairs(u.r, gap=0, cex=0.2) # looks good

## what about a meta-Clayton model?
## the parameter is chosen such that Kendall’s tau equals (the same) tau
copC <- onacopula("Clayton", list(copClayton@iTau(tau), 1:5))

## plot of the transformed data (Hering and Hofert (2014)) to see the
## deviations from uniformity
u.H <- htrafo(u, cop=copC) # transform the data
pairs(u.H, gap=0, cex=0.2) # clearly visible

## plot of the transformed data (Rosenblatt (1952)) to see the
## deviations from uniformity
u.R <- rtrafo(u, cop=copC) # transform the data
pairs(u.R, gap=0, cex=0.2) # clearly visible

## rtrafo() for elliptical:
fn <- fitCopula(normalCopula(dim=ncol(u)), u) # fit a Gauss copula
pairs(rtrafo(u, cop=fn@copula), gap=0, cex=0.2) # visible but not so clearly
if(copula:::doExtras()) {
  f.t <- fitCopula(tCopula(dim=ncol(u)), u)
tCop <- f.t@copula
} else {
tCop <- tCopula(param = 0.685, df = 7, dim=ncol(u))
}
u.Rt <- rtrafo(u, cop=tCop) # transform with a fitted t copula
pairs(u.Rt, gap=0, cex=0.2) # *not* clearly 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: 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 Rémiillard. 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

\texttt{indepTestSim(n, p, m = p, N = 1000, verbose = TRUE, print.every = NULL)}
\texttt{indepTest(x, d, alpha=0.05)}
\texttt{dependogram(test, pvalues = FALSE, print = FALSE)}

Arguments

\begin{itemize}
  \item \texttt{n} sample size when simulating the distribution of the test statistics under independence.
  \item \texttt{p} dimension of the data when simulating the distribution of the test statistics under independence.
  \item \texttt{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.
  \item \texttt{N} number of repetitions when simulating under independence.
  \item \texttt{print.every} is deprecated in favor of \texttt{verbose}.
  \item \texttt{verbose} a logical specifying if progress should be displayed via \texttt{txtProgressBar}.
  \item \texttt{x} data frame or data matrix containing realizations (one per line) of the random vector whose independence is to be tested.
  \item \texttt{d} object of class "\texttt{indepTestDist}" as returned by the function \texttt{indepTestSim()}. It can be regarded as the empirical distribution of the test statistics under independence.
  \item \texttt{alpha} significance level used in the computation of the critical values for the test statistics.
  \item \texttt{test} object of class "\texttt{indepTest}" as returned by \texttt{indepTest()}. \texttt{pvalues} logical indicating whether the dependogram should be drew from test statistics or the corresponding p-values.
  \item \texttt{print} logical indicating whether details should be printed.
\end{itemize}

Details

The current (C code) implementation of \texttt{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).

Value

The function \texttt{indepTestSim()} returns an object of class "\texttt{indepTestDist}" whose attributes are: \texttt{sample.size}, \texttt{data.dimension}, \texttt{max.card.subsets}, \texttt{number.repetions}, \texttt{subsets} (list of the subsets for which test statistics have been computed), \texttt{subsets.binary} (subsets in binary 'integer' notation), \texttt{dist.statistics.independence} (a N line matrix containing the values of the test statistics for each subset and each repetition) and \texttt{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 \textit{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


See Also

`serialIndepTest`, `multIndepTest`, `multSerialIndepTest`.

Examples

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

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

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

if(copula:::doExtras()) { ## not run, typically:
  print(system.time(
    d <- indepTestSim(100,5)
  ))
}

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

test <- indepTest(x,d)
## Let us see the results:
```
## 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".

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 $\hat{\theta}^G$, the corresponding Kendall’s tau is $\tau^G(\hat{\theta}^G)$ where $\tau^G(\theta) = (\theta - 1)/\theta$ 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 $\tau^{-1}$, 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).

Author(s)

Marius Hofert

References


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 <- copGumbeliTau(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.

**Author(s)**

Martin Maechler

**See Also**

the `interval` class documentation, notably its reference to more sophisticated interval classes available for R.
Examples

```
interval("[0, 1)")
```

```r
## Two ways to specify open interval borders:
identical(interval("[-1,1["),
          interval("(-1,1]"))
```

```r
## infinite:
interval("[0, Inf]")
```

```r
## arithmetic with scalars works:
4 + 2 * interval("[0, 1.5]") # -> [4, 7)
```

```r
## 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.

Author(s)

Martin Maechler

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 distribution function of the Kendall distribution of an Archimedean copula is defined as

\[ K(u) = P(C(U_1, U_2, \ldots, U_d) \leq u), \]

where \( u \in [0, 1] \), and the \( d \)-dimensional \((U_1, U_2, \ldots, U_d)\) is distributed according to the copula \( C \). Note that the random variable \( C(U_1, U_2, \ldots, U_d) \) is known as probability integral transform. Its distribution function \( K \) is equal to the identity if \( d = 1 \), but is non-trivial for \( d \geq 2 \).

\( \text{Kn}() \) computes the empirical Kendall distribution function, \( pK() \), \( qK() \), \( dK() \), and \( rK() \) provide the distribution function, quantile function, density, and random number generator, respectively, for the Kendall distribution of an Archimedean copula.

Usage

\[ \text{Kn}(u, x) \]
\[ pK(u, \text{cop}, d, \text{n.MC=0, log.p=FALSE}) \]
\[ qK(u, \text{cop}, d, \text{n.MC=0, method=c("default", "simple", "sort", "discrete", "monoH.FC"), u.grid, ...}) \]
\[ dK(u, \text{cop}, d, \text{n.MC=0, log.p=FALSE}) \]
\[ rK(n, \text{cop}, d) \]
Arguments

- **u**: evaluation point(s) (have to be in \([0, 1]\)).
- **x**: data (in the \(d\)-dimensional space) based on which the Kendall distribution function is estimated.
- **cop**: `acopula` with specified parameter, or (currently for \(rK\) only) a `outer_nacopula`.
- **d**: dimension (not used when `cop` 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. (2011b).
- **log.p**: logical; if TRUE, probabilities \(p\) are given as \(\log p\).
- **method**: string for the method to compute the quantile function of \(K\). Currently, one of
  - "default": The default method. Currently chooses `method="monoH.FC"` with \(u.grid=0:128/128\).
  - "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"`).
- **n**: sample size for \(rK\).

Details

For a completely monotone Archimedean generator \(\psi\),

\[
K(u) = \sum_{k=0}^{d-1} \frac{\psi^{(k)}(\psi^{-1}(u))}{k!}(-\psi^{-1}(u))^k, \quad u \in [0, 1];
\]

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, the Kendall distribution function, its quantile function, density, and random number generator.
Author(s)

Marius Hofert

References


See Also

`htrafo` or `emde` (where `K` is used).

Examples

tau <- 0.5
(theta <- copGumbel@iTau(tau)) # 2
d <- 20
(cop <- onacopulaL("Gumbel", list(theta,1:d)))

```r
# basic check empirical Kendall distribution function
set.seed(271)
n <- 1000
U <- rCopula(n, 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=c(0,1),
xlab=expression(italic(u)), ylab=expression(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=c(0,1))
# Note: Kn(0) -> 0 for n -> Inf

# compute Kendall distribution function
u <- seq(0,1, length.out = 255)
Ku <- pk(u, cop=cop@copula, d=d) # exact
Ku.MC <- pk(u, cop=cop@copula, d=d, n.MC=1000) # via Monte Carlo

# build sample from K
set.seed(1)
n <- 200
W <- rK(n, cop)```
Compute \( f(a) = \log(1 - \exp(-a)) \) respectively \( g(x) = \log(1 + \exp(x)) \) quickly numerically accurately.

Usage

\[
\begin{align*}
\log1mexp(a, \text{cutoff} = \log(2)) \\
\log1pexp(x, c0 = -37, c1 = 18, c2 = 33.3)
\end{align*}
\]
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 \(\log1p\exp\); 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

Author(s)

Martin Maechler, May 2002; \(\log1p\exp()\) in 2012.

References


Examples

```r
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)
```
108

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

<table>
<thead>
<tr>
<th>loss</th>
<th>LOSS and ALAE Insurance Data</th>
</tr>
</thead>
</table>

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

Examples
data(loss)
Description

\( \text{sinc}(x) \) 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

\[
\text{sinc}(x) \\
A..Z(x, \alpha, \text{I.}\alpha = 1 - \alpha)
\]

Arguments

- \( x \) numeric argument in \([0, \pi]\), typically a vector.
- \( \alpha \) parameter in \((0,1]\).
- \( \text{I.}\alpha \) 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

\[
\frac{\sin(\alpha x)^\alpha \sin((1 - \alpha)x)^{1-\alpha}}{\sin(x)}, \quad x \in \left[0, \pi\right],
\]

where \( \alpha \in (0,1] \) is alpha.

Author(s)

Martin Maechler

References


See Also

retstable internally makes use of these functions.
Examples

```R
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")
```

---

**Description**

Analog of the independence test based on the empirical copula process proposed by Christian Genest and Bruno Rémiillard (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**

```R
multIndepTest(x, d, m=length(d), N=1000, alpha=0.05, verbose = TRUE,
    print.every = NULL)
```

**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 $m \ll p$ especially when $p$ is large.
- **N**: number of bootstrap/permutation samples.
- **alpha**: significance level used in the computation of the critical values for the test statistics.
- **print.every**: is deprecated in favor of `verbose`.
- **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).
multIndepTest

References


See Also

*indeptest*, *serialIndepTest*, *multSerialIndepTest*, *dependogram*.

Examples

```r
## 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]

c <- normalCopula(0.3,dim=3)
x <- cbind(y,rCopula(n, c),rCopula(n, c))

x[,1] <- abs(x[,1]) * sign(x[,2] * 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]

d <- c(2,2,2,3,3)

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

## Display the dependogram
```
multSerialIndepTest

**Serial Independence Test for Multivariate Continuous Time Series Based on the Empirical Copula Process**

**Description**

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

**Usage**

```r
multSerialIndepTest(x, lag.max, m=lag.max+1, N=1000, alpha=0.05, 
verbose = TRUE, print.every = NULL)
```

**Arguments**

- `x`: data frame or data matrix containing realizations the multivariate continuous time series whose serial independence is to be tested.
- `lag.max`: maximum lag.
- `m`: maximum cardinality of the subsets of 'lags' for which a test statistic is to be computed. It makes sense to consider \( m \ll \text{lag.max+1} \) especially when \( \text{lag.max} \) is large.
- `N`: number of bootstrap/permutation samples.
- `alpha`: significance level used in the computation of the critical values for the test statistics.
- `print.every`: is deprecated in favor of `verbose`.
- `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 "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 \( \text{in} \) in the last reference) and global.statistic.pvalue (corresponding p-value).
References


See Also

serialIndepTest, indepTest, multIndepTest, dependogram

Examples

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

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

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

---

**Mvdc**

*Multivariate Distributions Constructed from Copulas*

**Description**

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

mvdc(copula, margins, paramMargins, marginsIdentical = FALSE,
       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 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 vector of the copula dimension or a matrix with number of columns being the copula dimension, 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

```r
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.
mvdc-class

See Also

ellipCopula, archmCopula; the classes mvdc and copula.

Examples

```r
## 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 $\theta$, which then serve as $\theta_0$ and $\theta_1$ for a three-dimensional fully nested Archimedean copula of the specified family. First, for each $\theta_0$, $n$ random variates $V_0$ are generated. Then, given the particular $\theta_0$ and the realizations $V_0$, $n$ random variates $V_{01}$ are generated for each $\theta_1$ 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 $V_0$s. For the submatrix that remains if the first column is removed, row i (for $\theta_{0i}$) contains the run times for the $V_{01i}$s for a particular $\theta_0$ and all the admissible $\theta_1$s.
nacopula-class

Author(s)

Marius Hofert, Martin Maechler

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

```r
## takes about 7 seconds: % so we rather test a much smaller set in R CMD check
## Not run: nacfrail.time(10000, "Gumbel", tau= c(0.05,(1:9)/10, 0.95))
```

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.

Author(s)

Martin Maechler
See Also

onacopula for building (outer) "nacopula" objects. For the class definition of the copula component, see acopula.

Examples

```r
## 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(0.2)
theta1 <- copFrank@iTau(0.5)
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 $d \times d$ 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 $\Theta$, where $\Theta[j,k] = \theta$ of the bivariate Archimedean copula $C(U_j, U_k)$. 
**nesdepth**

**Author(s)**

Martin Maechler

**See Also**

the class `nacopula` (with its `dim` method).

**Examples**

```r
## 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)])))
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
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

\[ C(\theta, c(i_1, \ldots, i_c), \text{list}(C(\ldots), \ldots, C(\ldots))) \].

Note that \( C() \) has (maximally) three arguments: the first is the copula parameter (vector) \( \theta \), the second a (possibly empty) vector of integer indices of components (for the comp slot in nacopulas), and finally a (possibly empty) list of child copulas, each specified with in the \( C(\ldots) \) notation themselves.
nacList a list of length 3 (or 2), with elements

1. theta: \( \theta \)
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.

Author(s)

Martin Maechler

References

Those of the Archimedean families, for example, copGumbel.

See Also

The class definitions "nacopula", "outer_nacopula", and "acopula".

Examples

```r
## 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(0.2)
theta1 <- copGumbel@iTau(0.5)
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", NAlist)
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( sapply(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, thetibase)`
Arguments

copbase: a "base" copula, that is, a copula of class `acopula`. Must be one of the predefined families.

thetabase: the univariate parameter $\theta$ 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`.

Author(s)

Marius Hofert

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

```r
## 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 <- onacopula(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),
```
```r
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))
```

---

**p2P**  
*Convert (Rho) Matrices to and From Parameter Vectors*

**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 $\Sigma$ which is called “Rho” as well, written (capital Greek $\rho$) as $P$ (and hence sometimes erroneously pronounced "Pee"). Note that getsigma() works for all elliptical copulas and uses p2P() for the “unstructured” case, dispstr = "un".

**Usage**

p2P(param, d)
P2p(P)
getsigma/copula

d
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().
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.

See Also

ellipCopula, tCopula, normalCopula.

Examples

```r
## display the definitions
p2P
P2p

param <- (2:7)/10
tC <- tCopula(param, dim = 4, dispstr="un", df = 3)
## consistency of the three functions:
P <- p2P(param, d=4)
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))))
```

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 $\chi^2$ distribution.
  "QQgamma" a Q-Q plot after a map to the gamma distribution.
  "PPchisq" a P-P plot after a map to the $\chi^2$ distribution.
  "PPgamma" a P-P plot after a map to the gamma distribution.
  "none" no points are plotted.
Note: These methods merely just set $g_1$ and $g_2$ 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.

Author(s)

Marius Hofert, Martin Mächler.

References


See Also

pairwiseCcop for the tools behind the scenes. demo(gof_graph) for examples.

Examples

```r
## 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), 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@I1(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., 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)
Description

Methods for function `persp` to draw perspective plots (of two dimensional distributions from package `copula`.

Usage

```R
## S4 method for signature 'copula'
persp(x, fun,
     n = 51, delta = 0,
     xlab = "x", ylab = "y", zlab = deparse(substitute(fun))[1],
     theta = -30, phi = 30, expand = 0.618,
     ticktype = "detail", ...)

## S4 method for signature 'mvdc'
persp(x, fun,
     xlim, ylim, nx = 51, ny = 51,
     xis = seq(xlim[1], xlim[2], length = nx),
     yis = seq(ylim[1], ylim[2], length = ny),
     xlab = "x", ylab = "y", zlab = deparse(substitute(fun))[1],
     theta = -30, phi = 30, expand = 0.618,
     ticktype = "detail", ...)
```

Arguments

- **x**: either a "**copula" or a "**mvdc" object.
- **fun**: the function to be plotted; typically `dCopula` or `pCopula`.
- **n**: (for "copula"): the number of points in both directions to do the plotting. The function `fun` will be evaluated on a grid of size $n \times n$.
- **delta**: a very small number in [0, 1/2), defaulting to zero. The x- and y-vectors will use range $[0 + \delta, 1 - \delta]$, i.e., $[0, 1]$ by default.
- **xlim, ylim** ("mvdc"): the range of the x or y variable, respectively.
- **nx, ny** ("mvdc"): the number of points in x- or y-direction, respectively. The function `fun` will be evaluated on a grid of size $nx \times ny$.
- **xis, yis** ("mvdc"): instead of specifying `xlim, ylim and nx, ny`, the numeric vectors (of length `nx` and `ny`) may be specified directly.
- **xlab, ylab, zlab, theta, phi, expand, ticktype, ...**: arguments for (the default method of) `persp()`, the ones enumerated here all with different defaults than there.

Value

invisibly (**invisible**) a list with components

- **x, y**: the numeric vectors, as passed to `persp.default`.
- **z**: the $nx \times ny$ matrix, 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

```r
persp(frankCopula(-0.8), dCopula)
persp(claytonCopula(2), pCopula, main = "CDF of claytonCopula(2)")
## example with negative tau :
(th! <- iTau(ahmCopula(), -0.1))
persp(ahmCopula(th!), dCopula)
persp(ahmCopula(th!), pCopula, ticktype = "simple")# no axis ticks

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

```r
plackettCopula(param)
```

Arguments

- `param` a numeric vector specifying the parameter values.

Value

A Plackett copula object of class "plackettCopula".

References


See Also

- ellipCopula
- archmCopula
Examples

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

---

Evaluation of (Nested) Archimedean Copulas

Description

For a (nested) Archimedean copula (object of class `nacopula`) \( x \), \( p\text{Copula}(u, \ x) \) (or also currently still \( \text{pnacopula}(x, \ u) \)) evaluates the copula \( x \) at the given vector or matrix \( u \).

Usage

```r
## S4 method for signature 'matrix,nacopula'
pCopula(u, copula, ...)
```

```r
## *Deprecated*:
pnacopula(x, u)
```

Arguments

- `copula, x` (nested) Archimedean copula of dimension \( d \), that is, an object of class `nacopula`, typically from `onacopula(\ldots)`.
- `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 \( \psi \) at \( u \) is given by

\[
C(u) = \psi(\psi^{-1}(u_1) + \ldots + \psi^{-1}(u_d)), \quad u \in [0, 1]^d.
\]

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

\( p\text{Copula}(u, \ copula) \) is a generic function with methods for all our copula classes, see \( p\text{Copula} \).

Author(s)

Marius Hofert, Martin Maechler.
Examples

```r
## 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**

```r
pobs(x, na.last = "keep",
     ties.method = , lower.tail = TRUE)
```

**Arguments**

- `x` - `n` by `d`-matrix of random variates to be converted to pseudo-observations.
- `na.last`, `ties.method` - strings, passed to `rank`; see there.
- `lower.tail` - logical which, if FALSE, returns the pseudo-observations when applying the empirical marginal survival functions.

**Details**

Given `n` realizations `\( x_i = (x_{i1}, \ldots, x_{id})^T, i \in \{1, \ldots, n\} \)` of a random vector `X`, the pseudo-observations are defined via `\( u_{ij} = r_{ij}/(n + 1) \)` for `i \in \{1, \ldots, n\}` and `j \in \{1, \ldots, d\}`, where `r_{ij}` denotes the rank of `\( x_{ij} \)` among all `\( x_{kj}, k \in \{1, \ldots, n\} \)`. The pseudo-observations can thus also be computed by component-wise applying the 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 \( \text{pobs}(, \text{lower.tail} = \text{FALSE}) \) simply returns \( 1-\text{pobs}() \).

**Value**

matrix of the same dimensions as \( x \) containing the pseudo-observations.

**Author(s)**

Marius Hofert

**Examples**

```r
## 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[1,5],u[1,10], xlab=expression(italic(U)[1]), ylab=expression(italic(U)[2]))

## All components: pairwise plot
pairs(u, gap=0, pch=".", labels=as.expression( sapply(1:d, function(j) bquote(italic(U[[j]])))) )
```

---

**polylog**

*Polylogarithm \( \text{Li}_s(z) \) and Debye Functions*

**Description**

Compute the polylogarithm function \( \text{Li}_s(z) \), initially defined as the power series,

\[
\text{Li}_s(z) = \sum_{k=1}^{\infty} \frac{z^k}{k^s},
\]

for \( |z| < 1 \), and then more generally (by analytic continuation) as

\[
\text{Li}_1(z) = -\log(1 - z),
\]

and

\[
\text{Li}_{s+1}(z) = \int_0^z \frac{\text{Li}_s(t)}{t} \, dt.
\]
Currently, mainly the case of negative integer \( s \) is well supported, as that is used for some of the Archimedean copula densities.

For \( s = 2 \), \( \text{Li}_2(z) \) is also called ‘dilogarithm’ or “Spence’s function”. The "default" method uses the \texttt{dilog} or \texttt{complex_dilog} function from package \texttt{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 \texttt{gsl} package functions \texttt{debye_1} and \texttt{debye_2} (which cannot deal with non-finite \( x \).)

**Usage**

```r
polylog(z, s, 
    method = c("default", "sum", "negI-s-Stirling", 
               "negI-s-Eulerian", "negI-s-asym-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 \( s \in \{0, -1, \ldots\} \)
- \( \text{method} \) a string specifying the algorithm to be used.
- \( \text{logarithm} \) logical specified to return \( \log(\text{Li}(.)) \) instead of \( \text{Li}(.) \)
- \( \text{is.log.z} \) logical; if TRUE, the specified \( z \) argument is really \( w = \log(z) \); that is, we compute \( \text{Li}_s(\exp(w)) \), and we typically have \( w < 0 \), or equivalently, \( z < 1 \).
- \( \text{is.logmlog} \) logical; if TRUE, the specified argument \( z \) is \( tw = \log(-w) = \log(-\log(z)) \) (where as above, \( w = \log(z) \)).
- \( \text{asymp.w.order} \) currently only default is implemented.
- \( n.\text{sum} \) for \text{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 \( \text{Li}_1(z) \):

\[
\text{Li}_1(z) = -\log(1 - z), \quad \text{Li}_0(z) = \frac{z}{1 - z}, \quad \text{Li}_{-1}(z) = \frac{z}{(1 - z)^2}, \quad \text{Li}_{-2}(z) = \frac{z(1 + z)}{(1 - z)^3},
\]

\[
\text{Li}_{-3}(z) = \frac{z(1 + 4z + z^2)}{(1 - z)^4}, \text{ etc.}
\]

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:
\[
\text{Li}_{-n}(z) = \left( z \frac{\partial}{\partial z} \right)^n \frac{z}{1 - z} = \sum_{k=0}^{\infty} k! S(n + 1, k + 1) \left( \frac{z}{1 - z} \right)^{k+1} \quad (n = 0, 1, 2, \ldots),
\]

where \( S(n, k) \) are the Stirling numbers of the second kind.

Equivalent formulae applicable to negative integer orders are (Wood 1992, § 6)...

\[
\text{Li}_{-n}(z) = \frac{1}{(1 - z)^{n+1}} \sum_{k=0}^{n-1} \binom{n}{k} z^{n-k} = \frac{\sum_{k=0}^{n-1} \binom{n}{k} z^k}{(1 - z)^{n+1}}, \quad (n = 1, 2, 3, \ldots),
\]

where \( \binom{n}{k} \) are the Eulerian numbers; see also Eulerian.

Value

numeric/complex vector as \( z \), or \( x \), respectively.

Author(s)

Martin Maechler

References


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

```r
## The dilogarithm, polylog(z, s = 2) = Li_2(.) -- mathematically defined on C \setminus [1, \text{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():
```

```r
```
**polynEval**

Evaluate Polynomials

```r
rli2 <- curve(polylog(x, z1), -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(LL, L)
all.equal(LLL, L)

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):
p.p <- palette()
local({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(1),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)
```
Description

Evaluate a univariate polynomial at \( x \) (typically a vector), that is, compute, for a given vector of coefficients \( \text{coef} \), the polynomial \( \text{coef}[1] + \text{coef}[2] \times x + \ldots + \text{coef}[p+1] \times x^p \).

Usage

\[
\text{polyneval(coef, x)}
\]

Arguments

- \( \text{coef} \) 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 \text{polynEval()} in package \text{sfsmisc}.

Value

numeric vector or array, with the same dimensions as \( x \), containing the polynomial values \( p(x) \).

Author(s)

Martin Maechler; the \( R \) version has been in package \text{sfsmisc} for ages.

See Also

For a much more sophisticated treatment of polynomials, use the \text{polynom} package (for example, evaluation can be done via \text{predict.polynomial}).

Examples

\[
\text{polyneval(c(1,-2,1), x = -2:7) \# (x - 1)^2} \\
\text{polyneval(c(0, 24, -50, 35, -10, 1),} \\
\quad \text{x = matrix(0:5, 2,3)) \# 5 zeros!}
\]

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 \text{printNacopula} explicitly allows to customize the printing behavior. Otherwise, the \text{show()} method calls \text{printNacopula} with default arguments only.
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

\texttt{prob(x, l, u)}

Arguments

\begin{itemize}
  \item \texttt{x} \hspace{1cm} \text{copula of dimension, that is, an object of class \texttt{Copula}.}
  \item \texttt{l, u} \hspace{1cm} \text{\textit{d}-dimensional, \textbf{numeric}, lower and upper hypercube boundaries, respectively, satisfying } 0 \leq l_i \leq u_i \leq 1 \text{, for } i \in 1,\ldots,d.
\end{itemize}

Value

A \textbf{numeric} in \([0, 1]\) which is the probability \(P(l_i < U_i \leq u_i)\).

Author(s)

Marius Hofert, Martin Maechler

See Also

\texttt{pcopula(\ldots)}.

Examples

\begin{verbatim}
## 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 \texttt{l} and upper
## point \texttt{u}.
1 <- c(.7,.8,.6)
u <- c(1,1,1)
prob(C3, 1, u)

## ditto for a bivariate normal copula with rho = 0.8 :
prob(normalCopula(0.8), c(.2,.4), c(.3,.6))
\end{verbatim}

\texttt{qqplot2} \hspace{1cm} \textit{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 \(1-\alpha\) confidence intervals.
qqplot2

Usage

```r
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 = "gray50"),
  CI.mtext = list(text = paste("Pointwise asymptotic ", 100 * (1 - alpha),
                      "% confidence intervals"), side = 4,
                      cex = 0.6 * par("cex.main"), adj = 0, col = "gray50"),
  main = expression(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()`.
Author(s)
Marius Hofert.

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 distribution)
qqplot2(x, qf=function(p) qchisq(p, df=df),
       main=as.expression(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=as.expression(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=expression(bold(Exp(1)~~"Q-Q Plot")))

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

DATE  a character vector specifying the date
INTC  daily log-return of the Intel stock
MSFT  daily log-return of the Microsoft stock
GE  daily log-return of the General Electric
References


Examples

data(rdj)

---

**Description**

Generating random variates of an exponentially tilted stable distribution of the form

\[ \tilde{S}(\alpha, 1, (\cos(\alpha \pi/2)V_0)^{1/\alpha}, V_0 1_{\{\alpha=1\}}, h 1_{\{\alpha \neq 1\}}; 1), \]

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 \(\psi_0\).
- `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.
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

$$\psi_{01}(t; V_0) = \left(1 - \left(1 - (1 - e^{-\theta_1}) \theta_0/\theta_1 \right) V_0 \right).$$

for the given realizations $V_0$ of Frank’s $F_0$ and the parameters $\theta_0, \theta_1 \in (0, \infty)$ such that $\theta_0 \leq \theta_1$. 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 $\text{rej} < V_0(1 - e^{-\theta_0})^{V_0-1}$ 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 $\theta_0 \leq \theta_1$. The parameter approx denotes the largest number of summands in the sum-representation of $V_{01}$ before the asymptotic

$$V_{01} = V_0^{1/\alpha} S(\alpha, 1, \cos^{1/\alpha}(\alpha \pi/2), 1_{\{\alpha=1\}; 1})$$

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 $\theta_0$, $\theta_1$ and $\alpha$ 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.

Author(s)

Marius Hofert, Martin Maechler

References


See Also

`rFFrank`, `rFJoe`, `rSibuya`, and `rnacopula`.

`rnacopula`

Examples

```r
## 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)
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

\[
(1 - (1 - \exp(-t)(1 - e^{-\theta₁}))^α)/(1 - e^{-\theta₀}),
\]

for Frank, or

\[
1 - (1 - \exp(-t))^α,
\]

for Joe, respectively, where \( \theta₀ \) and \( \theta₁ \) denote two parameters of Frank (that is, \( \theta₀, \theta₁ \in (0, \infty) \)) and Joe (that is, \( \theta₀, \theta₁ \in [1, \infty) \)) satisfying \( \theta₀ \leq \theta₁ \) and \( α = \theta₀/\theta₁ \).

### Usage

```
rfFrank(n, theta0, theta1, rej)
rFJoe(n, alpha)
```

### Arguments

- **n**: number of variates from \( F \).
- **theta0**: parameter \( \theta₀ \).
- **theta1**: parameter \( \theta₁ \).
- **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 \( α = \theta₀/\theta₁ \) 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 \( n \).
Sampling Logarithmic Distributions

Description

Generating random variates from a Log(p) distribution with probability mass function

\[ p_k = \frac{p^k}{-\log(1 - p)k}, \quad k \in \mathbb{N}, \]

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, \texttt{rlogR} is a pure-\texttt{R} implementation of \texttt{rlog}. However, \texttt{rlogR} is not as fast as \texttt{rlog} (the latter being implemented in C).

Value

A vector of positive \texttt{integers} of length \( n \) containing the generated random variates.

Author(s)

Marius Hofert, Martin Maechler
References


Examples

```r
## 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

```r
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*.

Author(s)

Martin Maechler, 10 Feb 2012

See Also

- *rnacopula* for generating d-dimensional observations from an (outer) *nacopula*, e.g., from the *result* of *rnacModel()*.
Examples

```r
## Implicitly tests the function {with validity of outer_nacopula ..}
set.seed(11)
for(i in 1:40) {
  m1 <- rnanModel("Gumbel", d=sample(20:25, 1), pr.comp = 0.3,
                   rtau0 = function() 0.25)
  m2 <- rnanModel("Joe", d=3, pr.comp = 0.1, order="each")
  mC <- rnanModel("Clayton", d=20, pr.comp = 0.3,
                   rtau0 = function() runif(1, 0.1, 0.5))
  mF <- rnanModel("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

```r
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 \( \mathcal{R} \) 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`. 
Author(s)

Marius Hofert, Martin Maechler

References


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

```r
## 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`.

**Author(s)**

Marius Hofert, Martin Maechler

**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 \( V_0 \sim F_0 \) (a Gamma(1/theta, 1) distribution)

\[
n \leftarrow 1000
\]

\[
V_0 \leftarrow \text{copClayton@V0}(n, \theta) \quad \text{copClayton@V0}(n, \theta)
\]

## Given these variates \( V_0 \), sample the child copula, that is, the bivariate
## nested Clayton copula with parameter \( \theta \)

\[
U_{23} \leftarrow \text{rnchild}(C_3@childCops[[1]], \theta, V_0)
\]

## Now build the three-dimensional vectors of random variates by hand

\[
U_1 \leftarrow \text{copClayton@psi}(\text{rexp(n)}/V_0, \theta)
u \leftarrow \text{cbind}(U_1, U_{23}u)
\]

## Plot the vectors of random variates from the three-dimensional nested
## Clayton copula

\[
\text{splom2}(U)
\]

### Description

Given a matrix of iid multivariate data from a meta-elliptical or meta-Archimedean model, \( \text{RSpobs()} \) computes pseudo-observations of the radial part \( R \) and the vector \( S \) 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

\[
\text{RSpobs}(x, \text{do.pobs = TRUE, method = c("ellip", "archm"), ...})
\]

### 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 \( \text{pobs()} \) 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 \( G_q \) as given in Genest, Hofert, G. Nešlehová (2014)); if provided, qQg() is used in the transformation for obtaining pseudo-observations of \( R \) and \( S \) (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 \( S \) (see the code for more details).
Details

The construction of the pseudo-observations 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)).

Author(s)

Marius Hofert

References


See Also

pobs() for computing the “classical” pseudo-observations.

Examples

```r
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) + rmvt(n, sigma=Sigma, df=nu) # multiv. t_nu data
## note: this is *wrong*: rmvt(n, mean=mu, sigma=Sigma, df=nu)

## compute pseudo-observations of the radial part and uniform distribution
## once for la), once for lb) below
RS.t <- RSpobs(X, method="ellip", qQ=qf, qt(p, df=nu)) # 'correct'
RS.norm <- RSpobs(X, method="ellip", qQ=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))

## la) Graphically testing the radial part

## la) Q-Q plot of R against the correct quantiles
qqplot2(RS.t$R, RS.norm$R, main="Q-Q plot of R against the correct quantiles")
```
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 $\delta \in \mathbb{R}$; see Nolan (2010) for the parameterization $k \in \{0, 1\}$. The case $\gamma = 0$ is understood as the unit jump at $\delta$.

Usage

```r
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 $\delta \in \mathbb{R}$.
- `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.

Author(s)

Diethelm Wuertz wrote `rstable` (now in package `stabledist`) for Rmetrics; Martin Maechler vectorized it (also in `alpha`,...), fixed it for $\alpha = 1, \beta \neq 0$ and sped it up.

References


See Also

rstable which also allows the 2-parametrization and provides further functionality for stable distributions.

Examples

```r
# Generate and plot a series of stable random variates
set.seed(1953)
r <- rstable(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 \( [\text{lower}, \text{upper}] \) if \( \text{sign}(f(\text{lower})) \) does not satisfy the requirements at the interval end points; see the ‘Details’ section.

Usage

```r
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 \( \text{lower} \) or \( \text{upper} \) endpoint, respectively.
- **Sig**: desired sign of \( f(\text{upper}) \), 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, \( S \) can be specified, typically as \( S := \pm 1 \), to require \( S = \text{sign}(f(x_0 + \epsilon)) \) at the solution. In that case, the search interval \([l, u] \) must be such that \( S \times f(l) \leq 0 \) and \( S \times f(u) \geq 0 \).

Otherwise, by default, when \( S = \text{NULL} \), the search interval \([l, u] \) must satisfy \( f(l) \times f(u) \leq 0 \).

In both cases, when the requirement is not satisfied, \texttt{safeUroot()} tries to enlarge the interval until the requirement is satisfied.

Value

A list with four components, \( \text{root}, fN\text{.root}, \text{iter} \) and \( \text{estim prec} \); see \texttt{uniroot}.

Author(s)

Martin Maechler (from Martin’s \texttt{R} package \texttt{nor1mix}).

See Also

\texttt{uniroot}.

Examples

\begin{verbatim}
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) )
\end{verbatim}
Serial Independence Test for Continuous Time Series Based on the Empirical Copula Process

Description

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

Usage

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

Arguments

- `n` length of the time series when simulating the distribution of the test statistics under serial independence.
- `lag.max` maximum lag.
- `m` maximum cardinality of the subsets of 'lags' for which a test statistic is to be computed. It makes sense to consider \( m \ll \text{lag.max+1} \) especially when \( \text{lag.max} \) is large.
- `N` number of repetitions when simulating under serial independence.
- `print.every` is deprecated in favor of `verbose`.
- `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.
serialIndepTest

Value

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

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

References


See Also

indeptest, multIndepTest, multSerialIndepTest, dependogram

Examples

```r
## 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},
```
## setTheta

### Specify the Parameter(s) of a Copula

#### Description

Set or change the parameter $\theta$ (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()`.

#### Usage

```r
setTheta(x, value, ...)  
# S4 method for signature 'acopula,ANY'
setTheta(x, value, na.ok = TRUE, noCheck = FALSE, ...)
# S4 method for signature 'copula,ANY'
setTheta(x, value, na.ok = TRUE, noCheck = FALSE, ...)
# S4 method for signature 'outer_nacopula,numeric'
setTheta(x, value, na.ok = TRUE, noCheck = FALSE, ...)
```

#### 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.)
- `...` 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.

#### Value

an R object of the same class as `x`, with the main parameter (vector) (often called `theta`) set to `value`.

---

```r
## we set lag.max=3. This may take a while...

d <- serialIndepTestSim(100,3)

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

test <- serialIndepTest(x,d)

## Let us see the results:

test

## Display the dependogram:

dependogram(test,print=TRUE)

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

Martin Maechler

Examples

myC <- setTheta(copClayton, 0.5)
myC

# 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)))
Description

The Sibuya distribution $Sib(\alpha)$ can be defined by its Laplace transform

$$1 - (1 - \exp(-t))^\alpha, \ t \in [0, \infty),$$

its distribution function

$$F(k) = 1 - (-1)^k \binom{\alpha - 1}{k} = 1 - \frac{1}{kB(k, 1 - \alpha)}, \ k \in \mathbb{N}$$

(where $B$ denotes the beta function) or its probability mass function

$$p_k = \binom{\alpha}{k} (-1)^{k-1}, \ k \in \mathbb{N},$$

where $\alpha \in (0, 1]$.  

$pSibuya$ evaluates the distribution function.  

$dSibuya$ evaluates the probability mass function.  

$rSibuya$ generates random variates from $Sib(\alpha)$ 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 $X_i \sim Sib(\alpha)$.

This probability mass function can be shown (see Hofert (2010, pp. 99)) to be

$$\sum_{j=1}^{n} \binom{n}{j} \binom{j \alpha}{k} (-1)^{k-j}, \ k \in \{n, n+1, \ldots\}.$$  

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.ctrl1 = 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 \( P(X \leq x) \), otherwise, \( P(X > x) \).
method character string specifying which computational method is to be applied. Implemented are:
"log" evaluates the logarithm of the sum

\[
\sum_{j=1}^{n} \binom{n}{j} \left(\frac{j\alpha}{x}\right)(-1)^{x-j}
\]

in a numerically stable way;
"direct" directly evaluates the sum;
"Rmpfr*" are as method="direct" but use high-precision arithmetic; "Rmpfr" and "Rmpfr0" return doubles 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 `dsSumSibuya` is numerically highly challenging, even already for small \( \alpha \) values (for example, \( n \geq 10 \)), 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 integers of length \( n \) containing the generated random variates.
- **dSibuya, pSibuya**: a vector of probabilities of the same length as \( x \).
- **dsSumSibuya**: a vector of probabilities, positive if and only if \( x \geq n \) and of the same length as \( x \) (or \( n \) if that is longer).

Author(s)

Marius Hofert, Martin Maechler
References


See Also

`rFJoe` and `rF01Joe` (where `rSibuya` is applied).

Examples

```r
## 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)
```

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:

```r
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", "SMN.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 })))
}

---

Scatterplot Matrix (splom) with Nice Variable Names
Description

A version of lattice’s splom function, particularly useful for visualizing multivariate data sampled from copulas, notably nested Archimedean ones.

Experimental We may replace the interface entirely, for example, to accept an "outer_nacopula".

Usage

splom2(data, varnames=NULL, Vname="U", xlab="", col.mat = NULL, bg.col.mat = NULL, ...)

Arguments

data numeric matrix or as.matrix(,)able.

varnames variable names, typically unspecified.

Vname character string to become the "base name" of the variables.

xlab x-axis label.

col.mat matrix of colors for the plot symbols (the default is the setting as obtained from trellis.par.get("plot.symbol")$col).

bg.col.mat matrix of colors for the background (the default is the setting as obtained from trellis.par.get("background")$col).

... further arguments, passed to splom().

Value

from splom(), an R object of class "trellis".

Author(s)

Martin Maechler, with a hint from Deepayan Sarkar; based on ideas from Marius Hofert.

Examples

## Create a 100 x 7 matrix of random variates from a t distribution
## with four degrees of freedom and plot the generated data
U7 <- matrix(rt(700, 4), 100, 7)
G <- splom2(U7)
G
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

```r
Stirling1(n, k)
Stirling2(n, k, method = c("lookup.or.store", "direct"))
Eulerian(n, k, method = c("lookup.or.store", "direct"))
```

```r
Stirling1.all(n)
Stirling2.all(n)
Eulerian.all(n)
```

Arguments

- **n**: positive integer (\( 0 \) is allowed for \( \text{Eulerian}() \)).
- **k**: integer in \( 0:n \).
- **method**: for \( \text{Eulerian}() \) and \( \text{Stirling2}() \), string specifying the method to be used. “direct” uses the explicit formula (which may suffer from some cancelation for “large” \( n \)).

Details

Eulerian numbers:
\[ A(n, k) = \text{the number of permutations of 1,2,...,n with exactly } k \text{ ascents (or exactly } k \text{ descents).} \]

Stirling numbers of the first kind:
\[ s(n,k) = (-1)^{n-k} \text{ times the number of permutations of 1,2,...,n with exactly } k \text{ cycles.} \]

Stirling numbers of the second kind:
\[ S_n^{(k)} \text{ is the number of ways of partitioning a set of n elements into } k \text{ non-empty subsets.} \]

Value

\[ A(n, k), s(n, k) \text{ or } S(n, k) = S_n^{(k)}, \text{ respectively.} \]

\( \text{Eulerian.all(n)} \) is the same as \( \text{sapply}(0:(n-1), \text{Eulerian}, n=n) \) (for \( n > 0 \)),
\( \text{Stirling1.all(n)} \) is the same as \( \text{sapply}(1:n, \text{Stirling1}, n=n) \), and
\( \text{Stirling2.all(n)} \) is the same as \( \text{sapply}(1:n, \text{Stirling2}, n=n) \), 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 \( n \geq 220 \).

Author(s)

Martin Maechler ("direct": May 1992)

References

Eulerians:

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: \( \text{http://dlmf.nist.gov/26.8} \)

Examples

\begin{align*}
\text{Stirling1}(7,2) \\
\text{Stirling2}(7,3)
\end{align*}

\begin{align*}
\text{Stirling1.all}(9) \\
\text{Stirling2.all}(9)
\end{align*}

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.

\text{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 \( \theta \) as large as \( \theta = 0.001 \).

\text{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 (= θ), we use Taylor series approximations of up to order 7,

\[
\tau_A(θ) = \frac{2}{9}θ\left(1 + \theta\left(\frac{1}{4} + \theta\left(\frac{1}{2} + \frac{2}{7}θ\right)\right)\right) + O(θ^6),
\]

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 (= θ), with τ values

for tauAMH: in [(5 - 8log2)/3, 1/3] = [-0.1817, 0.3333], of \(\tau_A(θ) = 1 - 2(θ + (1 - θ)^2 \log(1 - θ))/(3θ^2)\), numerically accurately, to at least around 12 decimal digits.

for tauJoe: in [-1,1].

Author(s)

Martin Maechler

See Also

acopula-families, and their class definition, "acopula".
Examples

```r
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 ~= 2/3*theta in the limit

curve(tauJoe, 1, 10)
curve(tauJoe, 0.2387, 10)# negative taus (*not* valid for Joe: no 2-monotone psi())
```

---

### Description

These data consist of log concentrations of 7 chemical elements in 655 water samples collected near Grand Junction, CO (from the Montrose quad-range of Western Colorado). Concentrations were measured for the following elements: Uranium (U), Lithium (Li), Cobalt (Co), Potassium (K), Cesium (Cs), Scandum (Sc), And Titanium (Ti).

### Usage

```r
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 Cobalt.
- **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


### Examples

```r
data(uranium)
```
# Index

*Topic **arith**
- Bernoulli, 25
- coeffG, 33
- interval, 101
- polylog, 134
- polynEval, 137
- Stirling, 167

*Topic **array**
- p2P, 124

*Topic **classes**
- acopula-class, 14
- archmCopula-class, 23
- copula-class, 40
- ellipCopula-class, 46
- evCopula-class, 58
- fgmCopula-class, 67
- fitCopula-class, 73
- indepCopula-class, 95
- interval-class, 102
- mvdc-class, 115
- nacopula-class, 117

*Topic **datasets**
- copFamilies, 36
- loss, 108
- rdj, 142
- SMI, 12, 164
- uranium, 170

*Topic **distribution**
- absdPsicMC, 12
- acR, 16
- archmCopula, 20
- beta.Blemqvist, 27
- cCopula, 31
- Copula, 38
- dDiag, 42
- dnacopula, 43
- ellipCopula, 44
- evCopula, 57
- fgmCopula, 66
- gnacopula, 81
- gofOtherTstat, 89
- gofTstat, 90
- gtrafo, 92
- indepCopula, 95
- K, 103
- Mvdc, 113
- opower, 122
- plackettCopula, 131
- pnacopula, 132
- prob, 139
- retstable, 143
- rf01FrankJoe, 144
- rFFrankJoe, 146
- rlog, 147
- rnacModel, 148
- rnacopula, 149
- rnchild, 150
- rstable1, 155
- Sibuya, 161
- tauAMH, 168

*Topic **goodness-of-fit**
- gofCopula, 83
- gofOtherTstat, 89
- gofTstat, 90

*Topic **hplot**
- .pairsCond, 10
- contour-methods, 34
- pairsRosenblatt, 125
- persp-methods, 129
- qqplot2, 140
- splom2, 165

*Topic **htest**
- An, 18
- evTestA, 59
- evTestC, 61
- evTestK, 62
- exchEVTest, 63
- exchTest, 65
INDEX

*Topic manip
   allComp, 18
   getAcop, 78
   p2P, 124
   setTheta, 160

*Topic math
   log1mexp, 106
   mathfun, 109

*Topic methods
   contour-methods, 34
   persp-methods, 129
   prob, 139
   show-methods, 161

*Topic models
   emde, 48
   emle, 50
   enacopula, 52
   estim.misc, 54
   fitCopula, 68
   fitMvdc, 74
   gofCopula, 83
   gofEVCopula, 87

*Topic multivariate
   pairsCond, 10
   acopula-class, 14
   An, 18
   archmCopula, 20
   assocMeasures, 24
   beta.Bloqvist, 27
   C.n, 29
   Copula, 38
   ellipCopula, 44
   evCopula, 57
   evTestA, 59
   evTestC, 61
   evTestK, 62
   exchEVTest, 63
   exchTest, 65
   fgMCopula, 66
   fitCopula, 68
   fitMvdc, 74
   generator, 76
   ggraph-tools, 79
   gnacopula, 81
   gofCopula, 83
   gofEVCopula, 87
   gofOtherTstat, 89
   gofTstat, 90
   indepTest, 96
   multIndepTest, 110
   multSerialIndepTest, 112
   serialIndepTest, 158

*Topic optimize
   safeUroot, 156

*Topic package
   copula-package, 4

*Topic print
   show-methods, 161

*Topic transformation
   gtrafo, 92

*Topic utilities
   allComp, 18
   interval, 101
   nacFrail.time, 116
   nacPairthetas, 118
   nesdepth, 119
   printNacopula, 138
   RSpobs, 152
   ac.classNames (getAcop), 78
   ac.longNames (getAcop), 78
   ac.objNames (getAcop), 78
   ac.shortNames (getAcop), 78
   emle, 53
   emle (emle), 50
   pairsCond, 10, 126-128
   %in%,numeric,interval-method
      (interval-class), 102

A, 19
A (generator), 76
A, asymCopula-method (generator), 76
A, galambosCopula-method (generator), 76
INDEX

A, gumbelCopula-method (generator), 76
A, huslerReissCopula-method (generator), 76
A, indepCopula-method (generator), 76
A, tawnCopula-method (generator), 76
A, tevCopula-method (generator), 76
A-methods (generator), 76
A...Z (math-fun), 109
absdPsiMC, 12, 93
acopula (acopula-class), 14
acopula-class, 14
acopula-families, 78
acopula-families (copFamilies), 36
acR, 16
Afun (generator), 76
AfunDer (generator), 76
allComp, 18
amhCopula (archmCopula), 20
amhCopula-class (archmCopula-class), 23
An, 18, 57, 60–62, 78, 88
Anfun (An), 18
archmCopula, 9, 20, 23, 24, 40, 41, 45, 57–59, 78, 95, 96, 115, 131
archmCopula-class, 23
array, 79, 80, 126
assocMeasures, 24
axis, 11
Bernoulli, 25
beta (beta.Blomqvist), 27
beta.Blomqvist, 27
beta.hat (beta.Blomqvist), 27
betan (beta.Blomqvist), 27
C.n, 29
cacopula (cCopula), 31
calibKendallsTau (assocMeasures), 24
calibSpearmansRho (assocMeasures), 24
cCopula, 31
cracter, 29, 33, 48, 53, 69, 78, 80–82, 84, 90, 99, 121, 126, 139, 141, 152
class, 85, 88, 102, 149
claytonCopula (archmCopula), 20
claytonCopula-class
(claytonCopula-class), 23
Cn (C.n), 29
cop, 69, 74
cop.fittedMV (fitMvdc), 74
coeffG, 33
complex_dilog, 135
contour, 34, 35
contour, copula-method (contour-methods), 34
contour, indepCopula-method (contour-methods), 34
contour, mvdc-method (contour-methods), 34
contour-methods, 34
copAMH, 16
copAMH (copFamilies), 36
copClayton, 16, 21, 22, 39
copClayton (copFamilies), 36
copFamilies, 36
copFrank, 16
copFrank (copFamilies), 36
copGumbel, 9, 15, 16, 22, 25, 39, 121, 150
copGumbel (copFamilies), 36
copJoe, 16
copJoe (copFamilies), 36
copula, 38, 39, 67, 71, 76, 92, 140, 160, 161
copula, 23, 24, 32, 34, 35, 39, 40, 47, 59, 68, 69, 73, 77, 84, 114, 115, 130, 131
Copula-class (copula-class), 40
copula-class, 40
copula-package, 4
cor, 55
dAdu (generator), 76
dAdu, galambosCopula-method (generator), 76
dAdu, gumbelCopula-method (generator), 76
dAdu, huslerReissCopula-method (generator), 76
dAdu, tawnCopula-method (generator), 76
dAdu, tevCopula-method (generator), 76
dAdu-methods (generator), 76
data.frame, 110
dCn (C.n), 29
dCopula, 9, 23, 34, 43, 130
dCopula (Copula), 38
dcopula (Copula), 38
dCopula, matrix, amhCopula-method (Copula), 38
dCopula, matrix, asymCopula-method (Copula), 38

copula, 9, 23, 34, 43, 130
dCopula (Copula), 38
dcopula (Copula), 38
dCopula, matrix, amhCopula-method (Copula), 38
dCopula, matrix, asymCopula-method (Copula), 38
<table>
<thead>
<tr>
<th>Function Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dCopula, matrix, asympExplicitCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, matrix, claytonCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, matrix, fgmCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, matrix, frankCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, matrix, gumbelCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, matrix, huslerReissCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, matrix, indepCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, matrix, joecCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, matrix, nacopula-method</td>
<td>43</td>
</tr>
<tr>
<td>dCopula, matrix, normalCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, matrix, plackettCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, matrix, tawnCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, matrix, tcopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, matrix, tevcopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, numeric, amhCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, numeric, asymCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, numeric, asympExplicitCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, numeric, claytonCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, numeric, fgmCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, numeric, frankCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, numeric, gumbelCopula-method</td>
<td>38</td>
</tr>
<tr>
<td>dCopula, numeric, huslerReissCopula-method</td>
<td>38</td>
</tr>
</tbody>
</table>

**Additional Functions**

- dDiag, 42
- debye1 (polylog), 134
- debye2 (polylog), 134
- debye_1, 135
- demo, 51, 54, 81, 117, 127, 128
- dependogram, 111, 113, 159
- dependogram (indepTest), 96
- digamma, 169
- dilog, 135
- dim, 18, 119, 120
- dim, copula-method (copula-class), 40
- dim, mvdc-method (mvdc-class), 115
- dim, nacopula-method (nacopula-class), 117
- diPsi (generator), 76
- diPsi, amhCopula-method (generator), 76
- diPsi, claytonCopula-method (generator), 76
- diPsi, frankCopula-method (generator), 76
- diPsi, gumbelCopula-method (generator), 76
- diPsi, joecopula-method (generator), 76
- diPsi-methods (generator), 76
- dK (K), 103
- dMvdc (Mvdc), 113
- dmvdc (Mvdc), 113
- dmvtn, 39
- dnacopula, 42, 43
- double, 167
- dSibuya (Sibuya), 161
- dsumSibuya, 33
- dsumSibuya (Sibuya), 161
INDEX

ebeta, 54, 100
ebeta (estim.misc), 54
edmle, 53, 54, 100
edmle (estim.misc), 54
edmcopula, 9, 22, 40, 41, 44, 47, 57, 70, 84, 92, 95, 115, 124, 125, 131
edmcopula-class, 46
dme, 48, 53, 54, 56, 94, 100, 105
dme, 50, 54, 56, 71, 100, 136
enacopula, 49, 51, 52, 53, 56, 81, 82, 100
evironment, 123
estim.misc, 54
etau, 54
Eulerian, 9, 26, 136
Eulerian (Stirling), 167
evCopula, 9, 19, 22, 24, 41, 57, 59–62, 87, 88, 95, 96
evCopula-class, 58
evTestA, 19, 59, 61, 62, 88
evTestC, 19, 59, 60, 61, 62, 88
evTestK, 19, 59–62, 62, 88
exchEVTest, 19, 63, 66
exchTest, 64, 65
F (n C n), 29
FALSE, 48
fgmCopula, 40, 41, 66, 66
fgmCopula-class, 67
findInterval, 104
fitCopula, 9, 45, 67, 68, 69, 73, 74, 76, 84, 86, 161
fitCopula-class, 73
fitdistr, 76
fitMvdc, 73, 74, 75, 116, 161
fitMvdc-class (fitCopula-class), 73
fittedMV-class (fitCopula-class), 73
formals, 114
format, interval-method
(interval-class), 102
frankCopula (archmCopula), 20
frankCopula-class (archmCopula-class), 23
function, 14, 15, 34, 80, 126, 130, 148, 152
galambosCopula, 57, 58
galambosCopula (evCopula), 57
galambosCopula-class (evCopula-class), 58
generator, 76
genFun (generator), 76
genFunDer1 (generator), 76
genFunDer2 (generator), 76
genInv (generator), 76
GenZBretz, 45
geta, 16, 37, 78
getSigma, 45
getsigma (p2P), 124
ggraph-tools, 79
gnacopula, 81
gofBTstat (gofOtherTstat), 89
gofCopula, 49, 64, 66, 71, 76, 82, 83, 88, 90, 91, 94
gofEVCopula, 19, 57, 59–62, 87
gofMB (gofCopula), 83
gofOtherTstat, 89
gofPB (gofCopula), 83
gofTstat, 82–84, 86, 90
gpviTest (ggraph-tools), 79
gtro, 83, 92
gumbelCopula, 57, 59
gumbelCopula (archmCopula), 20
gumbelCopula-class (archmCopula-class), 23
htrafo, 48, 49, 81–84, 105
htrafo (gtrafo), 92
huslerReissCopula, 57
huslerReissCopula (evCopula), 57
huslerReissCopula-class (evCopula-class), 58
image, 11
indepCopula, 21, 95, 95, 96
indepCopula-class, 95
indepTest, 79, 80, 96, 110, 111, 113, 158, 159
indepTestSim, 80
indepTestSim (indepTest), 96
initialize, acopula-method
(acopula-class), 14
initOpt, 99
integer, 18, 29, 50, 53, 80, 92, 104, 117, 145, 147, 151, 155, 162, 163
interval, 14, 15, 101, 101, 102, 103
interval-class, 102
invisible, 130, 141
iPsi (generator), 76
iPsi, archmCopula-method (generator), 76
iPsi, claytonCopula-method (generator), 76
iPsi, frankCopula-method (generator), 76
iPsi, gumbelCopula-method (generator), 76
iPsi, joeCopula-method (generator), 76
iPsi-methods (generator), 76
irho (assocMeasures), 24
irho, ANY-method (assocMeasures), 24
irho, claytonCopula-method (assocMeasures), 24
irho, copula-method (assocMeasures), 24
irho, elliptCopula-method (assocMeasures), 24
irho, fgmCopula-method (assocMeasures), 24
irho, frankCopula-method (assocMeasures), 24
irho, galambosCopula-method (assocMeasures), 24
irho, gumbelCopula-method (assocMeasures), 24
irho, huslerReissCopula-method (assocMeasures), 24
irho, jcopula-method (assocMeasures), 24
irho, tCopula-method (assocMeasures), 24
irho, tevCopula-method (assocMeasures), 24
irho-methods (assocMeasures), 24
iTau (assocMeasures), 24
iTau, acopula-method (assocMeasures), 24
iTau, amhCopula-method (assocMeasures), 24
iTau, ANY-method (assocMeasures), 24
iTau, claytonCopula-method (assocMeasures), 24
iTau, copula-method (assocMeasures), 24
iTau, elliptCopula-method (assocMeasures), 24
iTau, fgmCopula-method (assocMeasures), 24
iTau, frankCopula-method (assocMeasures), 24
iTau, galambosCopula-method (assocMeasures), 24
iTau, gumbelCopula-method (assocMeasures), 24
iTau, huslerReissCopula-method (assocMeasures), 24
iTau, jcopula-method (assocMeasures), 24
iTau, nacopula-method (assocMeasures), 24
iTau, normalCopula-method (assocMeasures), 24
iTau, plackettCopula-method (assocMeasures), 24
iTau, tawnCopula-method (assocMeasures), 24
iTau, tCopula-method (assocMeasures), 24
iTau, tevCopula-method (assocMeasures), 24
iTau-methods (assocMeasures), 24
joeCopula (archmCopula), 20
joeCopula-class (archmCopula-class), 23
K, 49, 81, 83, 92, 103, 105
kendallsTau (assocMeasures), 24
Kn (K), 103

lines, 141, 142
list, 11, 49–51, 56, 69, 73, 84, 114, 117, 120, 121, 125, 126, 128, 141, 153
log, 33, 42, 43, 77, 114
log1mexp, 106
log1pexp (log1mexp), 106
logical, 17, 80, 99, 102, 104, 133, 152, 163
loglik, 69, 74
loglik.fittedMV (fitMvdc), 74
loglikCopula (fitCopula), 68
loglikMvdc (fitMvdc), 74
loss, 108
1SMI (SMI.12), 164

math-fun, 109
matrix, 29, 79, 80, 89, 124–128, 152, 153, 164
maybeInterval-class (interval-class), 102
min, 102
mle, 50, 51, 54
mle2, 50, 51
mpfr, 33, 163
INDEX

mtext, 141, 142
multIndepTest, 98, 110, 113, 159
multSerialIndepTest, 98, 111, 112, 159
mvdc, 113
mvdc, 34, 35, 71, 73–76, 114–116, 130, 131
mvdc (Mvdc), 113
mtext, 177

NA, 135, 160
NA_real_, 41
nac2list (onacopula), 120
nacFrail.time, 116
nacopula, 16, 18, 24, 39, 41, 43, 118–121, 132, 138, 139, 148, 150, 151
nacopula (onacopula), 120
nacopula-class, 117
nPairThetas, 118
nesdepth, 119
nobs, 69
normalCopula, 45, 124, 125
normalCopula (ellipCopula), 44
normalCopula-class (ellipCopula-class), 46
NULL, 14, 92, 156

onacopula, 16, 117, 118, 120, 132, 149–151, 160
onacopulaL (onacopula), 120
opower, 122
optim, 50, 69, 70, 73, 75, 87
optimize, 48–51, 53, 55, 56
outer_nacopula, 18, 32, 42, 43, 48, 50, 53, 55, 81, 92, 104, 120, 121, 148–151, 160, 166
outer_nacopula-class (nacopula-class), 117

p.adjust.methods, 80
P2p (p2P), 124
p2P, 45, 124
pacR (acR), 16
pairs, 11, 126
pairsColList, 11
pairsColList (pairsRosenblatt), 125
pairsRosenblatt, 10, 12, 80, 81, 125
pairwiseCcop, 10, 11, 126, 128

pairwiseCcop (ggraph-tools), 79
pairwiseIndepTest (ggraph-tools), 79
par, 11
pCopula, 9, 23, 30, 34, 45, 130, 132, 140
pCopula (Copula), 38
pcopula (Copula), 38
pCopula, matrix, amhCopula-method (Copula), 38
pCopula, matrix, asymCopula-method (Copula), 38
pCopula, matrix, asymExplicitCopula-method (Copula), 38
pCopula, matrix, claytonCopula-method (Copula), 38
pCopula, matrix, fgmCopula-method (Copula), 38
pCopula, matrix, frankCopula-method (Copula), 38
pCopula, matrix, galambosCopula-method (Copula), 38
pCopula, matrix, gumbelCopula-method (Copula), 38
pCopula, matrix, huslerReissCopula-method (Copula), 38
pCopula, matrix, indepCopula-method (Copula), 38
pCopula, matrix, joeCopula-method (Copula), 38
pCopula, matrix, nacopula-method (pnacopula), 132
pCopula, matrix, normalCopula-method (Copula), 38
pCopula, matrix, plackettCopula-method (Copula), 38
pCopula, matrix, tawnCopula-method (Copula), 38
pCopula, matrix, tcopula-method (Copula), 38
pCopula, matrix, tevcopula-method (Copula), 38
pCopula, numeric, amhCopula-method (Copula), 38
pCopula, numeric, asymCopula-method (Copula), 38
pCopula, numeric, asymExplicitCopula-method (Copula), 38
pCopula, numeric, claytonCopula-method (Copula), 38
pCopula, numeric, fgmCopula-method
(Copula), 38
pCopula, numeric, frankCopula-method
(Copula), 38
pCopula, numeric, galambosCopula-method
(Copula), 38
pCopula, numeric, gumbelCopula-method
(Copula), 38
pCopula, numeric, huslerReissCopula-method
(Copula), 38
pCopula, numeric, indepCopula-method
(Copula), 38
pCopula, numeric, joeCopula-method
(Copula), 38
pCopula, numeric, nacopula-method
(nacopula), 132
pCopula, numeric, normalCopula-method
(Copula), 38
pCopula, numeric, plackettCopula-method
(Copula), 38
pCopula, numeric, tawnCopula-method
(Copula), 38
pCopula, numeric, tCopula-method
(Copula), 38
pCopula, numeric, tevCopula-method
(Copula), 38
pdf, 141, 142
persp, 130
persp, copula-method (persp-methods), 129
persp, mvdc-method (persp-methods), 129
persp-methods, 129
persp.default, 130
pK (K), 103
plackettCopula, 131, 131
plackettCopula-class (copula-class), 40
plot, 141, 142
pMvdc (Mvdc), 113
pmvnorm, 45
pmvt, 45
pnaCopula, 77, 132
pobs, 29, 30, 53, 69, 70, 74, 81, 92, 133, 152, 153
points, 126
polylog, 9, 134
polyn.eval, 138
polynEval, 137
predict.polynomial, 138
print, 161
print.default, 139
printNacopula, 117, 138
prob, 9, 139
prob, Copula-method (prob), 139
prob-methods (prob), 139
psi (generator), 76
psi, amhCopula-method (generator), 76
psi, claytonCopula-method (generator), 76
psi, frankCopula-method (generator), 76
psi, gumbelCopula-method (generator), 76
psi, joeCopula-method (generator), 76
psi-methods (generator), 76
pSibuya (Sibuya), 161
psiDabsMC (absdPsiMC), 12
psigamma, 168
pvitTest (gggraph-tools), 79
qacR (acR), 16
qK, 92
qK (K), 103
qqline, 126, 141, 142
qqplot2, 140
range, 35, 102, 130
rank, 133
cCopula, 9
cCopula (Copula), 38
cCopula (Copula), 38
cCopula, numeric, amhCopula-method
(Copula), 38
cCopula, numeric, asymCopula-method
(Copula), 38
cCopula, numeric, asymExplicitCopula-method
(Copula), 38
cCopula, numeric, claytonCopula-method
(Copula), 38
cCopula, numeric, evCopula-method
(Copula), 38
cCopula, numeric, fgmCopula-method
(Copula), 38
cCopula, numeric, frankCopula-method
(Copula), 38
cCopula, numeric, galambosCopula-method
(Copula), 38
cCopula, numeric, gumbelCopula-method
(Copula), 38
cCopula, numeric, huslerReissCopula-method
(Copula), 38
INDEX

rCopula, numeric, indepCopula-method (Copula), 38
rCopula, numeric, joeCopula-method (Copula), 38
rCopula, numeric, nacopula-method (Copula), 38
rCopula, numeric, normalCopula-method (Copula), 38
rCopula, numeric, plackettCopula-method (Copula), 38
rCopula, numeric, tCopula-method (Copula), 38
rdj, 142
retstable, 109, 143
retstableR (retstable), 143
rF01Frank, 146, 147
rF01Frank (rF01FrankJoe), 144
rF01FrankJoe, 144
rF01Joe, 147, 164
rFFrank, 145
rFFrank (rFFrankJoe), 146
rFFrankJoe, 146
rFJoe, 145, 164
rFJoe (rFFrankJoe), 146
rho, 9, 136
rho (assocMeasures), 24
rho, acopula-method (assocMeasures), 24
rho, amhCopula-method (assocMeasures), 24
rho, ANY-method (assocMeasures), 24
rho, claytonCopula-method (assocMeasures), 24
rho, copula-method (assocMeasures), 24
rho, evCopula-method (assocMeasures), 24
rho, fgmCopula-method (assocMeasures), 24
rho, frankCopula-method (assocMeasures), 24
rho, galambosCopula-method (assocMeasures), 24
rho, gumbelCopula-method (assocMeasures), 24
rho, huslerReissCopula-method (assocMeasures), 24
rho, indepCopula-method (assocMeasures), 24
rho, nacopula-method (assocMeasures), 24
rho, normalCopula-method (assocMeasures), 24
rho, plackettCopula-method (assocMeasures), 24
rho, tawncopula-method (assocMeasures), 24
rho, tCopula-method (assocMeasures), 24
rho, tevCopula-method (assocMeasures), 24
rho-methods (assocMeasures), 24
rK (K), 103
rlog, 147
rlogR (rlog), 147
rMvdc (Mvdc), 113
rmvdc (Mvdc), 113
rnacModel, 148, 150
rnacopula, 9, 145, 147, 148, 149, 151
rnchild, 149, 150, 150
rownames, 164
rSibuya, 145–147
rSibuya (Sibuya), 161
rSibuyaR (Sibuya), 161
RSpobs, 152
rstable, 155, 156
rstable (rstable1), 155
rstable1, 144, 154
rtrafo, 32, 81–84
rtrafo (gtrafo), 92
rug, 141, 142

safeUroot, 55, 56, 156
serialIndepTest, 98, 111–113, 158
serialIndepTestSim (serialIndepTest), 158
setClassUnion, 41, 102
setTheta, 160
setTheta, acopula, ANY-method (setTheta), 160
setTheta, copula, ANY-method (setTheta), 160
setTheta, ellipCopula, ANY-method (setTheta), 160
setTheta, outer_nacopula, numeric-method (setTheta), 160
show, 138, 161
show, acopula-method (acopula-class), 14
show, copula-method (show-methods), 161
show, fitCopula-method (show-methods), 161
show, fitMvdc-method (show-methods), 161
show, interval-method (interval-class), 102
INDEX

index

tCopula (ellipCopula), 44
   tCopula-class (ellipCopula-class), 46
   tevCopula, 57
   tevCopula (evCopula), 57
   tevCopula-class (evCopula-class), 58
   trellis.par.get, 166
   txtProgressBar, 82, 84, 87, 97, 110, 112, 158

   unirroot, 17, 56, 104, 156, 157
   uranium, 170

   vcov, 69, 74
   vcov.fittedMV (fitMvdc), 74
   vector, 124, 125, 127, 128, 153
   vignettes, 9