Package ‘copula’

March 5, 2015

Version 0.999-13

Date SDate: 2015-03-05 17:50:12 +0100 (Thu, 05. Mar 2015) $,
subversion $Revision: 1156 $ <-- 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 (>= 2.15.2)

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

Suggests MASS, KernSmooth, sfsmisc, scatterplot3d, Rmpfr, bbmle,
partitions, polynom, rugarch, colorspace, mvnormtest, tseries,
zoo

SuggestsNote the last line packages partitions, ..., zoo are only used
in demos

Enhances nor1mix

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
Collate

- AllClass.R
- Classes.R
- Copula.R
- Auxiliaries.R
- aux-acopula.R
- special-func.R
- cop_objects.R
- nacopula.R
- derCdfPdf.R
- amhCopula.R
- amhExpr.R
- An.R
- archmCopula.R
- asymCopula.R
- asymExplicitCopula.R
- claytonCopula.R
- claytonExpr.R
- ellipCopula.R
- empCop.R
- empPsi.R
- acR.R
- estimation.R
- evCopula.R
- evTests.R
- exchTests.R
- fgmCopula.R
- fitCopula.R
- fitMvdc.R
- frankCopula.R
- frankExpr.R
- galambosCopula.R
- galambosExpr.math.R
- galambosExpr.R
- ggraph-tools.R
- pairsRosenblatt.R
- device.R
- gofTrafos.R
- gofEVTests.R
- gofTests.R
- graphics.R
- gumbelCopula.R
- gumbelExpr.R
- huslerReissCopula.R
- huslerReissExpr.R
- indepCopula.R
- indepTests.R
- joeCopula.R
- K.R
- logseries.R
- mvdc.R
- normalCopula.R
- plackettCopula.R
- plackettExpr.R
- rstable1.R
- safeUroot.R
- schlatherCopula.R
- stable.R
- timing.R
- trafos.R
- tCopula.R
- tawnCopula.R
- tawnExpr.R
- tevCopula.R
- wrapper.R
- zzz.R

Encoding UTF-8

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

NeedsCompilation yes

Repository CRAN

Date/Publication 2015-03-05 22:13:21

R topics documented:

copula-package ................................................................. 4
.pairsCond ................................................................. 6
absdPsiMC ................................................................. 8
acopula-class ............................................................... 9
acR ................................................................. 12
allComp ................................................................. 13
An ................................................................. 14
archmCopula ................................................................. 16
archmCopula-class .......................................................... 18
assocMeasures ............................................................... 19
Bernoulli ................................................................. 21
beta.Blomqvist .............................................................. 22
C.n ................................................................. 24
cCopula ................................................................. 27
contour-methods ............................................................ 28
copFamilies ................................................................. 30
Copula ................................................................. 33
copula-class ................................................................. 35
dDiag ................................................................. 36
device ................................................................. 37
dnacopula ................................................................. 38
ellipCopula ................................................................. 38
ellipCopula-class ........................................................... 40
emde ................................................................. 41
R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>emle</td>
<td>44</td>
</tr>
<tr>
<td>enacopula</td>
<td>47</td>
</tr>
<tr>
<td>estim.misc</td>
<td>49</td>
</tr>
<tr>
<td>evCopula</td>
<td>51</td>
</tr>
<tr>
<td>evCopula-class</td>
<td>53</td>
</tr>
<tr>
<td>evTestA</td>
<td>54</td>
</tr>
<tr>
<td>evTestC</td>
<td>55</td>
</tr>
<tr>
<td>evTestK</td>
<td>57</td>
</tr>
<tr>
<td>exchEVTest</td>
<td>58</td>
</tr>
<tr>
<td>exchTest</td>
<td>59</td>
</tr>
<tr>
<td>fgmCopula</td>
<td>61</td>
</tr>
<tr>
<td>fgmCopula-class</td>
<td>62</td>
</tr>
<tr>
<td>fitCopula</td>
<td>63</td>
</tr>
<tr>
<td>fitCopula-class</td>
<td>68</td>
</tr>
<tr>
<td>fitMvdc</td>
<td>69</td>
</tr>
<tr>
<td>generator</td>
<td>71</td>
</tr>
<tr>
<td>getAcop</td>
<td>73</td>
</tr>
<tr>
<td>ggraph-tools</td>
<td>74</td>
</tr>
<tr>
<td>gnacopula</td>
<td>76</td>
</tr>
<tr>
<td>gofCopula</td>
<td>78</td>
</tr>
<tr>
<td>gofEVCopula</td>
<td>81</td>
</tr>
<tr>
<td>gofOtherTstat</td>
<td>83</td>
</tr>
<tr>
<td>gofTstat</td>
<td>84</td>
</tr>
<tr>
<td>gtrafo</td>
<td>86</td>
</tr>
<tr>
<td>indepCopula</td>
<td>89</td>
</tr>
<tr>
<td>indepCopula-class</td>
<td>90</td>
</tr>
<tr>
<td>indepTest</td>
<td>91</td>
</tr>
<tr>
<td>initOpt</td>
<td>94</td>
</tr>
<tr>
<td>interval</td>
<td>96</td>
</tr>
<tr>
<td>interval-class</td>
<td>97</td>
</tr>
<tr>
<td>K</td>
<td>98</td>
</tr>
<tr>
<td>log1mexp</td>
<td>101</td>
</tr>
<tr>
<td>loss</td>
<td>102</td>
</tr>
<tr>
<td>math-fun</td>
<td>103</td>
</tr>
<tr>
<td>multIndepTest</td>
<td>104</td>
</tr>
<tr>
<td>multSerialIndepTest</td>
<td>106</td>
</tr>
<tr>
<td>Mvdc</td>
<td>108</td>
</tr>
<tr>
<td>mvdc-class</td>
<td>110</td>
</tr>
<tr>
<td>nacFrail.time</td>
<td>111</td>
</tr>
<tr>
<td>nacopula-class</td>
<td>112</td>
</tr>
<tr>
<td>nacPairthetas</td>
<td>113</td>
</tr>
<tr>
<td>nesdepth</td>
<td>114</td>
</tr>
<tr>
<td>onacopula</td>
<td>115</td>
</tr>
<tr>
<td>opower</td>
<td>117</td>
</tr>
<tr>
<td>p2P</td>
<td>119</td>
</tr>
<tr>
<td>pairsRosenblatt</td>
<td>120</td>
</tr>
<tr>
<td>persp-methods</td>
<td>124</td>
</tr>
<tr>
<td>plackettCopula</td>
<td>126</td>
</tr>
</tbody>
</table>
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

Package: copula
Type: Package
License: GPL (>= 3)
Depends: methods
The **copula** package provides

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

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

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

Further information is available in the following vignettes:

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

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

**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 are

e.example(fCopula)## fitting Copulas
e.example(fitMvdc) ## fitting multivariate distributions via Copulas
e.example(nacopula) ## nested Archimedean Copulas

e.example(idepenTest) ## Testing for Independence
e.example(serIndepTest) ## 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 pairwiseCcop, 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),
```
.pairsCond

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

Description

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

Usage

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

Arguments

t
  numeric vector of evaluation points.

family
  Archimedean family (name or object).

theta
  parameter value.

degree
  order \( d \) of the derivative.

n.MC
  Monte Carlo sample size.

method
  different methods:
  "log": evaluates the logarithm of the sum involved in the Monte Carlo approximation in a numerically stable way;
  "direct": directly evaluates the sum;
  "pois.direct": interprets the sum in terms of the density of a Poisson distribution and evaluates this density directly;
  "pois": as for method="pois" but evaluates the logarithm of the Poisson density in a numerically stable way.

log
  if TRUE the logarithm of absdPsi is returned.

is.log.t
  if TRUE the argument \( t \) contains the logarithm of the “mathematical” \( t \), i.e., conceptually, \( \psi(t, \ast) = \psi(\log(t), \ast, \text{is.log.t=TRUE}) \), where the latter may potentially be numerically accurate, e.g., for \( t = 10^{500} \), where as the former would just return \( \psi(Inf, \ast) = 0 \).

Details

The absolute value of the \( d \)th derivative of the Laplace-Stieltjes transform \( \psi = LS[F] \) can be approximated via

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

where \( V_k \sim F, \quad k \in \{1, \ldots, N\} \). This approximation is used where \( d = \text{degree} \) and \( N = \text{n.MC} \). Note that this is comparably fast even if \( t \) contains many evaluation points, since the random variates \( V_k \sim F, \quad k \in \{1, \ldots, N\} \) 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(absolutePsiMC)$method), function(MM)
          absolutePsiMC(780, family="Gumbel", method = MM,
          theta=2, degree=10, n.MC=10000, log=TRUE, is.log.t = TRUE))

## not really better, yet...
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 \( \psi(t) = e^{-t} \)) 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 \((-1)^\text{degree} \times \text{absdPsi}\)). 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.

absdIPSi: a function computing the absolute value of the derivative of the generator inverse \( \text{iPSi}() \) 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 \text{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 the each of the arguments.

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

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

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

V0: A function which samples n random variates from the distribution \( F \) with Laplace-Stieltjes transform \( \psi \) 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 \( \theta_0 = \text{theta0}, \theta_1 = \text{theta1} \).
acopula-class

\(dv01\): Similar to \(dv0\) with the difference being that this function computes the probability mass or density function for the Laplace-Stieltjes transform

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

corresponding to the distribution function \(F_{01}\).

Arguments are the evaluation point(s) \(x\), the value(s) \(V0\), and the parameters \(theta0\) and \(theta1\). As for \(dv0\), the optional argument \(log\) can be specified (defaults to FALSE). Note that if \(x\) is a vector, \(V0\) must either have length one (in which case \(V0\) is the same for every component of \(x\)) or \(V0\) must be of the same length as \(x\) (in which case the components of \(V0\) correspond to the ones of \(x\)).

\(tau, iTau\): Compute Kendall’s tau of the bivariate Archimedean copula with generator \(\psi\) as a function of \(theta\), respectively, \(theta\) as a function of Kendall’s tau.

\(lambdaL, lambdaU, lambdaLInv, lambdaUInv\): Compute the lower (upper) tail-dependence coefficient of the bivariate Archimedean copula with generator \(\psi\) 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
doacopula()

# Information and structure of Clayton copulas
acopCopula
```
12

str(copClayton)

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

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

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

---

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

### Description

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

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

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

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:"*dm, 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:dim(\( x \)), whereas its “inner” component copulas will each contain a subset of those indices only.

Usage

```r
allComp(x)
```

Arguments

\[ x \]

an \( R \) object inheriting from class nacopula.
An integer vector of indices $j$ of all components $u_j$ as described in the description above.

Author(s)

Martin Maechler

Examples

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

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

```r
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**

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

**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)
curve(An.biv(rCopula(1000, gumbelCopula(2)), x), lty=2, add=TRUE, col=2)
curve(An.biv(rCopula(1000, gumbelCopula(1.33)), x), lty=2, add=TRUE, col=3)

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

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

## A trivariate example
x <- rCopula(1000, gumbelCopula(4, dim = 3))
```
archmCopula

Construction of Archimedean Copula Class Object

Description

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

Usage

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

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

Arguments

family a character string specifying the family of an Archimedean copula. Currently supported families are "clayton", "frank", "amh", "gumbel", and "joe".
param number (numeric) specifying the copula parameter.
dim the dimension of the copula.
... further arguments, passed to the individual creator functions (claytonCopula(), etc).

use.indepC a string specifying if the independence copula indepCopula, should be returned in the case where the parameter \( \theta \), 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".

u <- matrix(runif(300), 100, 3)
w <- u / apply(u, 1, sum)
r <- An(x, w)

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

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

For the mathematical definitions of the given Archimedean families, see copClayton.

For \(d = 2\), i.e. \(\dim = 2\), the AMH, Clayton and Frank copulas allow to model negative Kendall’s \(\tau\) (\(\tau\)) behavior via negative \(\theta\), 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. \(\dim > 2\), 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

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, -10, 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 with OUT message
Archimedean copula 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

expdist: 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.
assocMeasures

Methods

dCopula signature(copula = "claytonCopula"): ...
pCopula signature(copula = "claytonCopula"): ...
rCopula signature(copula = "claytonCopula"): ...
dCopula signature(copula = "frankCopula"): ...
pCopula signature(copula = "frankCopula"): ...
rCopula signature(copula = "frankCopula"): ...
dCopula signature(copula = "gumbelCopula"): ...
pCopula signature(copula = "gumbelCopula"): ...
rCopula signature(copula = "gumbelCopula"): ...
dCopula signature(copula = "amhCopula"): ...
pCopula signature(copula = "amhCopula"): ...
rCopula signature(copula = "amhCopula"): ...
dCopula signature(copula = "joeCopula"): ...
pCopula signature(copula = "joeCopula"): ...
rCopula signature(copula = "joeCopula"): ...

Extends

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

Note

"gumbelCopula" is also of class "evCopula".

See Also

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

________

assocMeasures    Dependence Measures for Bivariate Copulas

Description

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

tau (copula, ...)  
rho (copula, ...)  
tailIndex(copula, ...)  
iTau (copula, tau, ...)  
iRho (copula, rho, ...)  

Arguments

copula an R object of class "copula" (or also "acopula" or "ncopula"; 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

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),  
           ...)
Bernoulli

\[
c(\text{copGumbel@lambda(3), copGumbel@lambdaU(3)), check.attributes=FALSE),
\]

\[
\text{all.equal}(i\text{Tau (gumbel.cop, 0.681), copGumbel@i\text{Tau(0.681))})
\]

## let us compute the sample versions
x <- rCopula(200, gumbel.cop)
cor(x, method = "kendall")
cor(x, method = "spearman")

## compare with the true parameter value 3
i\text{Tau(gumbel.cop, cor(x, method="kendall")}[1,2])
i\text{Rho(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**

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

**Arguments**

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

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

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

- **verbose**
  - (for "A-T"): logical indicating if the intermediate results of the algorithm should be printed.
Sample and Population Version of Blomqvist's Beta for Archimedean Copulas

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

```r
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 \(\text{betan}\): \((n \times d)\)-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


See Also

- `acopula`

Examples

```r
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)), .8)

## AMH:
for(i in seq_along(d.set))
  curve(Vectorize(beta.,"theta")(copAMH, x, d = d.set[i]), 0, .999999,
       main = "Blomqvist's beta(.) for AMH",
       xlab = 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)
```
## 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"))
```
\( dC_n(u, U, j.\text{ind}=1:d, b=1/\sqrt{\text{nrow}(U)}, \ldots) \)

\( C_n(x, w) \#\# \text{deprecated! use } C.n(w, U=\text{pobs}(x)) \text{ instead!} \)

**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**: (and \(x\) for \(Cn()\)) an \((n, d)\)-matrix 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 \(\text{pobs}()\).
- **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 \(\text{sum}(\ldots)/(n+\text{offset})\); defaults to zero.
- **method**: character string indicating which method is applied to compute the empirical CDF or copula. \(\text{method}="C"\) uses a an implementation in C, \(\text{method}="R"\) 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,

\[
C_n(u) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(\hat{U}_i \leq u),
\]

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 \(j.\text{ind}\)), 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

```r
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=expression(italic(C)[n](italic(u),...,italic(u))))
pC <- function(x) pCopula(do.call(cbind, rep(list(x), d)), cop)
curve(pC, 1ty=2, add=TRUE)
legend("topleft", 1ty=1:2, bty="n", inset=0.02,
       legend=c(expression(italic(C)[n]), expression(italic(C)))))
```
## cCopula

### Conditional Copula Function

#### Description

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

#### Usage

```r
cCopula(u, copula, j.ind=ncol(u), n.MC=0, log=FALSE)
```

#### Arguments

- **u**: \( n \times d \)-matrix; the conditioning is done on the values in the first \( d - 1 \) columns.
- **copula**: 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.
- **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

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)
U <- 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

---

contour-methods

Methods for Function 'contour' in Package 'copula'

Description

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

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

```r
## 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 = TRUE, ...)
```

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 x 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 \([0+\delta, 1-\delta]\), 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 x 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.
- `...`: 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):
copFamilies

Specific Archimedean Copula Families ("acopula" Objects)

Description

Specific Archimedean families ("acopula" objects) implemented in the package copula. These families are “classical” as from p. 116 of Nelsen (2007). More 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

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

  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.
copFamilies

**copFrank**: Archimedean family of Frank with parametric generator

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

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

**copGumbel**: Archimedean family of Gumbel with parametric generator

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

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

**copJoe**: Archimedean family of Joe with parametric generator

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

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

Note that staying within one of these Archimedean families, all of them can be nested if two (generic) generator parameters \(\theta_0, \theta_1\) 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@parInterval

## 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@lambda(u), xlim = c(1, 10), ylim = c(0,1), col = 4)

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

## Plot psi() and tau() - and properties of all families ----
## The copula families currently provided:
(famNms <- ls("package:copula", patt="cop[A-Z]")

op <- par(mfrow = c(length(famNms), 2),
       mar = .6 + c(2,1.4,1,1), mgp = c(1.1, 0.4, 0))
for(nm in famNms) {
  Cf <- get(nm)
  thet <- Cf@iTau(0.3)
  curve(Cf@psi(x, theta = thet), 0, 5,
       xlab = 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@parInterval
  Iu <- pmin(10, I[2])
  curve(Cf@tau(x), I[1], Iu, col = 3,
       xlab = bquote(theta %in% .(format(I))), ylab = "",
       main = substitute(list(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?

```r
C2@copula@lambda(theta)
C2@copula@lambdaU(theta)
```

## Generate n pairs of random variates from this copula

```r
U <- rncopula(n = 1000, C2)
```

## and plot the generated pairs of random variates

```r
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

```r
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.
- **...**: further optional arguments for some methods, e.g., method.

#### Details

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

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

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

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

References


See Also

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

Examples

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

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

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

## a 3-dimensional clayton copula
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 <- copClaytondacopula(v, theta = 2)
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.lowbnd: lower bounds for the parameters, of class "numeric".
- param.upbnd: 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)
```

---

### dDiag

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

**Description**

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

**Usage**

```r
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

```r
th. <- c(0.1, 0.2, 0.5, 0.8, 1.4, 2., 5.)
curve(dDiag(x, cop=onacopulaL("Clayton", list(th.[1], 1:3))), 0, 1,
   n=1000, ylab="dDiag(x, *", main="Diagonal densities of Clayton")
   abline(h=0, lty=3)
   for(j in 2:length(th.))
      curve(dDiag(x, cop=onacopulaL("Clayton", list(th.[j], 1:3))), add=TRUE,
          col=j, n=1000)
legend("topleft", do.call(expression, lapply(th., function(th)
               substitute(theta == TH, list(TH=th)))))
   lty=1, col=seq_along(th.), bty="n")
```
character a string containing a font embedding command.
    """ do not embed fonts.

... additional arguments passed to dev.off().

Value

invisible().

Author(s)

Marius Hofert.

See Also

dev.off() for closing a device, embedFonts() for font embedding. sfsmisc’s pdf.end() for another approach.

Examples

## typical usage

doPDF <- !dev.interactive(orNone=TRUE)
if(doPDF) pdf(file=(file <- "crop_device.pdf"), width=6, height=6)
plot()
if(doPDF) dev.off.pdf(file)

---

## S4 method for signature 'matrix,nacopula'

dCopula(u, copula, log=FALSE, ...)

## Deprecated:
dnacopula(x, u, log=FALSE, ...)
... optional arguments passed to the copula’s dacopula function (slot), such as nMC (non-negative integer) for possible Monte Carlo evaluation (see dacopula in acopula).

Details

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

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

Value

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

Note

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

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))

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

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

### Description

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

### Usage

```r
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`.
- **...**: currently nothing.

### Value

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

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

See Also

p2P(), and getSigma() for construction and extraction of the “Rho” (P) or Sigma matrix of (generalized) correlations.
archmCopula, fitCopula.

Examples

```r
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")) {
  ## 3d scatter plot of 1000 random observations
  scatterplot3d(rCopula(1000, norm.cop))
  scatterplot3d(rCopula(1000, t.cop))
}

## 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")
```

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 be 'ex', 'ar1', 'toep', or 'un'.
dimension: Object of class "numeric", dimension of the copula.
parameters: Object of class "numeric", parameter value.
param.names: Object of class "character", parameter names.
param.lowbnd: Object of class "numeric", parameter lower bound.
param.upbnd: Object of class "numeric", parameter upper bound.
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**: A $n \times d$-matrix of (pseudo-)observations (each value in $[0, 1]$) from the copula, where $n$ denotes the sample size and $d$ the dimension.
- **cop**: The 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**: A bivariate vector denoting the interval where optimization takes place. The default is computed as described in Hofert et al. (2013).
- **include.K**: A 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**: A logical indicating whether the distance function to be optimized is reparametrized (the default); see the code for more details.
- **...**: Additional arguments passed to optimize.

Details

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

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

Value

A list as returned by optimize, including the minimum distance estimator.
Author(s)
Marius Hofert

References

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

Examples
```r
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)
  list(value=val, error=val-theta, utime.ms=1000*run.time[[1]])
}
(res <- sapply(meths, fun, u=U, cop=cop, theta=theta))
```

---

**emle**

*Maximum Likelihood Estimators for (Nested) Archimedean Copulas*

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

Usage
```r
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 \( \mathcal{R} \) function \texttt{mle()} from package \texttt{stats4}, whereas the default, "optimize" uses \texttt{optimize} via the \( \mathcal{R} \) function \texttt{mle2()} from package \texttt{bbmle}.
method      only when optimizer is NULL or "optimize", the method to be used for \texttt{optim}.
interval    bivariate vector denoting the interval where optimization takes place. The default is computed as described in Hofert et al. (2012).
start       \texttt{list} of initial values, passed through.
...         additional parameters passed to \texttt{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 \texttt{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 \( \mathcal{R} \) object of class "\texttt{mle2}" (and thus useful for obtaining confidence intervals) with the (simulated) maximum likelihood estimator.

.emle \texttt{list} as returned by \texttt{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
```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)

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

## Profile likelihood plot [using S4 methods from bbmle/stats4]:
pfm <- profile(emle)
ci <- confint(pfm, level=0.95)
ci
stopifnot(ci[1] <= theta, theta <= ci[2])
plot(pfm) # |z| against theta, |z| = sqrt(deviance)
plot(pfm, absVal=FALSE, # z against theta
     show.points=TRUE) # showing how it's interpolated

## and show the true theta:
abline(v=theta, col="lightgray", lwd=2, lty=2)
axis(1, pos = 0, at=theta, label=expression(theta[0]))

## Plot of the log-likelihood, MLE and conf.int.:
logL <- function(x) -efm@minuslogl(x)
    # == sum(copGumbel@dacopula(U, theta=x, log=TRUE))
logL. <- Vectorize(logL)
I <- c(cop@copula@iTau(0.1), cop@copula@iTau(0.4))
curve(logL., from=I[1], to=I[2], xlab=quote(theta),
ylab="log-likelihood",
```
enacopula

Estimation Procedures for (Nested) Archimedean Copulas

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

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

Arguments
u  $ n \times d$-matrix of (pseudo-)observations (each value in $[0,1]$) from the copula to be estimated, where $n$ denotes the sample size and $d$ the dimension. Consider applying the function pobs first in order to obtain $u$.
cop outer_nacopula to be estimated (currently only Archimedean copulas are provided).
method a character string specifying the estimation method to be used, which has to be one (or a unique abbreviation) of
"mle" maximum likelihood estimator (MLE) computed via .emle.
"smle" simulated maximum likelihood estimator (SMLE) computed with the function .emle, where n.MC gives the Monte Carlo sample size.
"dmle" MLE based on the diagonal (DMLE); see edmle.
"mde.chisq.CvM" minimum distance estimator based on the chi-square distribution and Cramér-von Mises distance; see emde.
"mde.chisq.KS" minimum distance estimation based on the chi-square 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 \texttt{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

\texttt{n.MC} only for method = "smle": \texttt{integer}, sample size for simulated maximum likelihood estimation.

\texttt{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".

\texttt{xargs} list of additional arguments for the chosen estimation method.

\texttt{...} additional arguments passed to \texttt{optimize}.

Details

\texttt{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.

Author(s)

Marius Hofert, Martin Maechler.

References


See Also

\texttt{emle} which returns an object of "mle" providing useful methods not available for other estimators. \texttt{demo(opC-demo)} and \texttt{demo(GIG-demo)} for examples of two-parameter families. \texttt{edmle} for the diagonal maximum likelihood estimator. \texttt{emde} for the minimum distance estimators. \texttt{etau} for the estimators based on Kendall’s tau. \texttt{ebeta} for the estimator based on Blomqvist’s beta.
Examples

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

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

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

fun <- function(method, u, cop, theta) {
    run.time <- system.time(val <- enacopula(u, cop=cop, method=method))
    list(value=val, error=val-theta, utime=100*run.time[[1]])
}
t(res <- sapply(meths, fun, u=U, cop=cop, theta=theta))
```

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

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

Arguments

- **u**  \(n \times d\)-matrix of (pseudo-)observations (each value in \([0,1]\)) from the copula, where \(n\) denotes the sample size and \(d\) the dimension.
- **cop**  `outer_nacopula` to be estimated (currently only Archimedean copulas are provided).
- **interval**  bivariate vector denoting the interval where optimization takes place. The default is computed as described in Hofert et al. (2013).
- **method**  a character string specifying the method (only for `etau`), which has to be one (or a unique abbreviation) of

  - “tau.mean” method-of-moments-like estimator based on the average of pairwise sample versions of Kendall’s tau;
estim.misc

"theta.mean" average of the method-of-moments-like Kendall's tau estimators.

warn logical indicating if warnings are printed:
edmle() for the family of "Gumbel" if the diagonal maximum-likelihood estimator is smaller than 1.
etau() for the family of "AMH" if tau is outside \([0, 1/3]\) and in general if at least one of the computed pairwise sample versions of Kendall's tau is negative.

... additional arguments passed to cor (for etau), to optimize (for edmle), or to safeUroot (for ebeta).

Details

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

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

For etau, the method="tau.mean" means that the average of sample versions of Kendall's tau are computed first and then the parameter is determined such that the population version of Kendall's tau matches this average (if possible); the method="theta.mean" stands for first computing all pairwise Kendall's tau estimators and then returning the mean of these estimators.

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

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

Value

ebeta the return value of safeUroot (that is, typically almost the same as the value of uniroot) giving the Blomqvist beta estimator.
edmle list as returned by optimize, including the diagonal maximum likelihood estimator.
etau method-of-moments-like estimator based on Kendall's tau for the chosen method.

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()))
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=bquote(tau), xlab=bquote(rho), 
    main= expression(tau( tevCopula(rho) )))
```
Description

Class evCopula is the virtual (mother) class of all extreme-value copulas. There currently are five subclasses, "galambosCopula", "huslerReissCopula", "tawnCopula", "tevcCopula", 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 "tevcCopula" 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")
evTestA(uC)
```

---

**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**

```r
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?
evTestC(rCopula(200, gumbelCopula(3)))
evTestC(rCopula(200, claytonCopula(3)))

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

_Bivariate Test of Extreme-Value Dependence Based on Kendall’s 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**

```
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**

`evTestC`, `evTestA`, `evCopula`, `gofEVCopula`, `an`.

**Examples**

```r
c <- 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))
```
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

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))
```

---

**exchTest**

*Test of Exchangeability for a Bivariate Copula*

**Description**
Test for assessing the exchangeability of the underlying bivariate copula based on the empirical copula. The test statistics are defined in the first two references. Approximate p-values for the test statistics are obtained by means of a multiplier technique.

**Usage**
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


See Also

exchEVT, gofCopula.

Examples

```r
## 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)
  u <- copula::mrt(u, v, m=0)
  v <- copula::mrt(v, u, m=0)
  copula::rcopula(n, cop, u, v)
}
```
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()
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
```

---

`fgmCopula-class`  
*Class "fgmCopula"*

Description

Multivariate Multiparameter Farlie-Gumbel-Morgenstern Copula.

Objects from the Class

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

Slots

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

---

**fitCopula**

*Estimation of the Parameters in Copula Models*

**Description**

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

**Usage**

```r
copula(x, 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), "ita" (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 \text{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 \text{optim}(\*, control=optim.control).

optim.method

the method for \text{optim}().

estimate.variance

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

Details

There are methods for \text{vcov()}, \text{coef()}, \text{logLik()}, and \text{nobs()}.

Value

\text{loglikCopula()} returns the log likelihood evaluated at the given value of "\text{param}".

The return value of \text{fitCopula()} is an object of class "\text{fitCopula}" (see there), containing slots (among others!)

\text{estimate} the estimate of the parameters.

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

copula the fitted copula.

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

\text{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 \text{fitCopula()} is used and if \text{start} is not assigned a value, estimates obtained from method "ita" are used as initial values in the optimization.

If methods "ita" or "irho" are used in \text{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 \text{pobs()}).
For the t copula with df.fixed=FALSE (see ellipCopula()), the methods "itau" and "irho" cannot be used in fitCopula(). For the methods "m1" and "mpl", 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 "mpl".

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 "m1". 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 <- ifelse(Xtras) 200 else 64

set.seed(7) # for reproducibility
x <- rCopula(n, gumbel.cop)## "true" observations
u <- pobs(x) # pseudo-observations
## Inverting Kendall's tau

```r
fit.tau <- fitCopula(gumbel.cop, u, method="itau")
fit.tau
```

## Inverting Spearman's rho

```r
fit.rho <- fitCopula(gumbel.cop, u, method="irho")
fit.rho
```

## Maximum pseudo-likelihood

```r
fit.mpl <- fitCopula(gumbel.cop, u, method="mpl")
fit.mpl
```

## Maximum likelihood

```r
fit.ml <- fitCopula(gumbel.cop, x, method="ml")
fit.ml
```

---

## A multiparameter example

```r
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")

fit.tau <- fitCopula(normal.cop.toep, u, method="itau")
fit.tau

fit.rho <- fitCopula(normal.cop.toep, u, method="irho")
fit.rho

fit.mpl <- fitCopula(normal.cop.toep, u, method="mpl")
fit.mpl

fit.ml <- fitCopula(normal.cop.toep, x, method="ml")
fit.ml
```
```r
## 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, x, 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$counts <- y@fitting.stats$counts <- NULL
            all.equal(x,y, ...)
        }
    stopifnot(all.eqCop(tc.ml , tc.ml, tolerance=.005),
              all.eqCop(tc.mpl, tc.mpl, tolerance=.005))

    ## same t copula but with df.fixed=TRUE (--> use same data!)
    (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.F.ml <- fitCopula(tc.f, x, method="ml", start=c(0,0,0)))
    (tc.F.ml <- fitCopula(tc.f, x, method="ml"))  # without 'start'
    stopifnot(all.eqCop(tc.F.ml,tc.F.ml, tolerance= 4e-4))
    ## the (estimated, asymptotic) var-cov matrix:
    vcov(tc.F.ml)

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

### fitCopula-class

**Classes of Fitted Multivariate Models: Copula, Mvdc**

#### Description

Classes and summary methods related to copula model fitting.

#### Objects from the Class

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

#### Slots

The “mother class”, "fittedMV" has the slots
- `estimate`: numeric, the estimated parameters.
- `var.est`: numeric, variance matrix estimate of the parameter estimator. See note below.
- `loglik`: numeric, log likelihood evaluated at the maximizer.
- `nsample`: numeric, integer representing the sample size.
- `fitting.stats`: a `list`, currently containing the numeric convergence code from `optim`, the counts, message, and all the control arguments explicitly passed to `optim`.

In addition, the "fitCopula" class has a slot
- `copula`: the fitted copula, of class "copula".

whereas the "fitMvdc" has
- `mvdc`: the fitted distribution, of class "mvdc".

#### Extends

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

#### Methods

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

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

#### References

Estimation of Multivariate Models Defined via Copulas

Description

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

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

Usage

```r
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)
```

 Arguments

- `param`: a vector of parameter values. When specifying parameters for `mvdc` objects, the parameters must be ordered with the marginals first and the copula parameters last. When the `mvdc` object has `marginsIdentical = TRUE`, only the parameters of one marginal must be specified.
- `x`: a data matrix.
- `mvdc`: a "mvdc" object.
- `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(11)
x <- rmvdc(10000, gMvd2)
## with identical margins:
gMvd1 <- 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 <- fitMvdc(x, gMvd2, start = c(1,1, 2),
                  hideWarnings=FALSE) ## <- show warnings here
## 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

```r
generator
generator

### psi

`psi(copula, s)`

### iPsi

`iPsi(copula, u, ...)`

### diPsi

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

### A

`A(copula, w)`

### dAdu

`dAdu(copula, w)`
```
Arguments

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

Details

- **psi** and **iPsi** are, respectively, the generator function \(\psi\) and its inverse \(\psi^{-1}\) for an Archimedean copula, see **pnacopula** for definition and more details.
- **diPsi** computes (currently only the first two) derivatives of **iPsi** \((= \psi^{-1})\).
- **A()**, the “Pickands dependence function”, can be seen as the generator function of an extreme-value copula. For instance, in the bivariate case, we have the following result (see, e.g., Gudendorf and Segers 2009):

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

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

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


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

getAcop(family, check=TRUE)

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

Arguments

family either a character string, the short or longer form of the Archimedean family name (for example, "Clayton" or simply "C": see the acopula-families documentation), or an acopula family object, or an object inheriting from class archmCopula.
check logical indicating whether the class of the return value should be checked to be "acopula".

Value

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

ggetAcop("Gumbel")

## different ways of getting the same "acopula" family object:
stopifnot(# Joe (three ways):
identical(getAcop("J"), getAcop("Joe")),

73
Computations for Graphical GOF Test via Pairwise Rosenblatt Transforms

Description

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

pairwiseCop() 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
pairwiseCop(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 pairwiseCop()). 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 pairwiseCop().
- **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 \(cu \cdot u\) with \(cu[u(i,j)]\) containing \(C(u_i \mid u_j)\) 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

```r
## demo(gof_graph)
```
gnacopula() conducts a goodness-of-fit test for the given \( (H_0) \)-copula \( \text{cop} \) based on the \( \text{data u} \).

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

Usage

\[
gnacopula(u, \text{cop}, n\text{.bootstrap},
\text{estim.method} = \text{eval(formals(enacopula)$method)},
\text{include.K=TRUE}, n.MC=0, \text{trafo=c("Hering.Hofert", "Rosenblatt")},
\text{method}=\text{eval(formals(goftstat)$method)}, \text{verbose=TRUE}, ...)
\]

Arguments

\( u \) \( \times d \)-matrix of values in \([0,1]\); should be (pseudo-/copula-)observations from the copula to be tested. Consider applying the function \text{pobs}() first in order to obtain \( u \).

\( \text{cop} \) \( H_0 \)-"outer_nacopula" with specified parameters to be tested for (currently only Archimedean copulas are provided).

\( n\text{.bootstrap} \) positive integer specifying the number of bootstrap replicates.

\( \text{estim.method} \) character string determining the estimation method; see \text{enacopula}(). We currently only recommend the default "mle" (or maybe "smle").

\( \text{include.K} \) logical indicating whether the last component, involving the Kendall distribution function \( K() \), is used in the transformation \text{htrafo}() of Hering and Hofert (2011). Note that this only applies to \text{trafo="Hering.Hofert"}.

\( n.MC \) parameter \( n.MC \) for \text{htrafo}() (and thus for \( K() \)) if \( \text{trafo="Hering.Hofert"} \) and for \text{rtrafo}() if \( \text{trafo="Rosenblatt"} \).

\( \text{trafo} \) 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 \text{htrafo}().

"Hering.Hofert" for the multivariate transformation of Rosenblatt (1952); see \text{rtrafo}().

\( \text{method} \) a character string specifying the goodness-of-fit test statistic to be used; see \text{gofstat}().

\( \text{verbose} \) if TRUE, the progress of the bootstrap is displayed via \text{txtProgressBar}.

\( \ldots \) additional arguments passed to \text{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 `estim.method` is used. Note that a finite-sample correction is made when computing p-values; see `gofCopula()` for details.

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

Value

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

- `p.value`: the bootstrapped p-value;
- `statistic`: the value of the test statistic computed for the data `u`;
- `data.name`: the name of `u`;
- `method`: a `character` describing the goodness-of-fit test applied;
- `estimator`: the estimator computed for the data `u`;
- `bootStats`: a list with component `estimator` containing the estimators for all bootstrap replicates 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 refer-
ces) 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 prinicipal 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", "AnChi2", 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 is a character string specifying the estimation method to be used to estimate the
dependence parameter(s); see `fitCopula()`.

simulation is 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 is 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` depend-
ing on the `optim.method`.

trafo.method is a string specifying the transformation to \( U[0, 1]^d \); either "none" or one of "rtrafo", see `rtrafo`, or "htrafo", see `htrafo`.

`trafoArgs` is 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 ex-
changeable, "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 asymptoti-
cally 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

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

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)

## Not run:
## A two-dimensional data example
```
x <- rCopula(200, claytonCopula(3))

(tau. <- cor(x, method="kendall")[1,2]) # around 0.5 -- 0.6
## Does the Gumbel family seem to be a good choice?
thG <- iTau(gumbelCopula(), tau.)
gofCopula(gumbelCopula(thG), x)
# SnC: really s..l...o...w... --- SnB is *EVEN* slower
gofCopula(gumbelCopula(thG), x, method = "SnC")
## What about the Clayton family?
thC <- iTau(claytonCopula(), tau.)
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 <- rCopula(200, tCopula(c(0.5, 0.6, 0.7), dim = 3, dispstr = "un"))

## Does the Clayton family seem to be a good choice?
gofCopula(gumbelCopula1(1, dim = 3), x)
## What about the t copula?
t.copula <- 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(gumbelCopula1(1, dim = 3), x, estim.method="itau")
gofCopula(t.copula, x, estim.method="itau")

## The same using the multiplier approach

gofCopula(gumbelCopula1(1, dim = 3), x, simulation="mult")
gofCopula(t.copula, x, simulation="mult")

## End(Not run)

---

**gofEVCopula**

*Goodness-of-fit Tests for Bivariate Extreme-Value Copulas*

**Description**

Goodness-of-fit tests for extreme-value copulas based on the empirical process comparing one of the two nonparameteric rank-based estimator of the Pickands dependence function studied in Genest and Segers (2009) with a parametric estimate of the Pickands dependence function derived under the null hypothesis. The test statistic is the Cramer-von Mises functional Sn defined in Equation (5) of Genest, Kojadinovic, G. Nešlehová, and Yan (2010). Approximate p-values for the test statistic are obtained using a parametric bootstrap.
Usage

gofEVCopula(copula, x, N = 1000, method = "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), "itau" (inversion of Kendall’s tau) or “irho” (inversion of Spearman’s rho).
estimator specifies which nonparametric rank-based estimator of the unknown Pickands dependence function to use; can be either "CFG" (Caperaa-Fougeres-Genest) or "Pickands".
m number of points of the uniform grid on [0,1] used to compute the test statistic numerically.
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
## 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="ita")
gofEVCopula(gumbelCopula(1), x, method="irho")

## The same with different extreme-value copulas
gofEVCopula(galambosCopula(1), x)
gofEVCopula(galambosCopula(1), x, method="ita")
gofEVCopula(galambosCopula(1), x, method="irho")

gofEVCopula(huslerReissCopula(1), x)
gofEVCopula(huslerReissCopula(1), x, method="ita")
gofEVCopula(huslerReissCopula(1), x, method="irho")

gofEVCopula(tevCopula(0, df.fixed=TRUE), x)
gofEVCopula(tevCopula(0, df.fixed=TRUE), x, method="ita")
gofEVCopula(tevCopula(0, df.fixed=TRUE), x, method="irho")

## End(Not run)

gofOtherTstat | Various Goodness-of-fit Test Statistics
---|---

**Description**

gofBTstat() computes supposedly Beta distributed test statistics for checking uniformity of u on the unit sphere.

**Usage**
gofBTstat(u)

**Arguments**

- **u**: \((n, d)\)-matrix of values whose rows supposedly follow a uniform distribution on the unit sphere in \(\mathbb{R}^d\).
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(plain("Beta")*(s1,s2)~~
                                      bold("-Q-Q Plot")), list(s1=k/2, s2=(d-k)/2))), side=3, cex=1.3, line=1.1, xpd=NA))
qqplot(1, Bmat=Bmat) # k=1
qqplot(3, Bmat=Bmat) # k=3
```

---

**gofTstat**

*Goodness-of-fit Test Statistics*

**Description**

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

**Usage**

         useR = FALSE, ...)

gofTstat

Arguments

u  \( n \times d \)-matrix of values in \([0, 1]\), supposedly independent uniform observations in the hypercube, that is, \( U_i \sim U[0, 1]^d \), i.i.d., for \( i \in \{1, \ldots, n\} \).

method  a character string specifying the goodness-of-fit test statistic to be used, which has to be one (or a unique abbreviation) of

- "Sn" for computing the test statistic \( S_n \) from Genest, Rémillard, Beaudoin (2009).
- "Snb" for computing the test statistic \( S_n^{(B)} \) from Genest, Rémillard, Beaudoin (2009).
- "SnC" for computing the test statistic \( S_n^{(C)} \) from Genest et al. (2009).
- "AnChisq" Anderson-Darling test statistic for computing (supposedly) \( U[0, 1] \)-distributed (under \( H_0 \)) random variates via the distribution function of chi-square distribution with \( d \) degrees of freedom. To be more precise, the Anderson-Darling test statistic of the variates

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

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

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

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).

userR  logical indicating whether an R or C implementation is used.

...  additional arguments passed for computing the different test statistics.

Details

This function should be used with care. The different test statistics were implemented (partly) for different purposes and goodness-of-fit tests and should be used only with knowledge about such (see the references for more details).

Value

The value of the test statistic, a numeric.

Author(s)

Marius Hofert, Martin Maechler.
References


See Also
gofCopula() for goodness-of-fit tests where (some of) these test statistics are used.

Examples

```r
## generate data
cop <- archmCopula("Gumbel", param=1Tau(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
```

### gtrafo

*GOF Testing Transformations for Archimedean Copulas*

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

```r
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**: A \(n \times d\)-matrix with values in \([0, 1]\). If `inverse=FALSE` (the default), \(u\) contains (pseudo-/copula-)observations from the copula \(\text{cop}\) 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 \((\text{cop})\) 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

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

This formula is either evaluated with the exact derivatives or, if \(n.MC\) is positive, via Monte Carlo; see `absdPsiMC`. Rosenblatt (1952) showed that \(U' \sim U[0,1]^m\), where \(U'_1 = U_1, U'_2 = C(U_2 | U_1), \ldots, U'_m = C(U_m | U_1, \ldots, U_{m-1})\).

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

The inverse transformation (\(\text{inverse=TRUE}\)) 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

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

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 (include.K=TRUE) or not (include.K=FALSE).

Value

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

\(\text{rtrafo()}\) 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 \(\text{emde}\) where \(\text{htrafo}\) is applied.

Examples

\[
\text{tau} \leftarrow 0.5 \\
(\text{theta} \leftarrow \text{copGumbel@iTau}(\text{tau})) \# 2 \\
(\text{copG} \leftarrow \text{onacopula}("Gumbel", \text{list(\text{theta}, 1:5)))} \# d = 5
\]

\[
\text{set.seed}(1) \\
\text{n} \leftarrow 1000 \\
\text{x} \leftarrow \text{rnacopula}(\text{n, copG}) \\
\text{x} \leftarrow \text{qnorm(x)} \# x now follows a meta-Gumbel model with \(N(0, 1)\) marginals \\
\text{u} \leftarrow \text{pobs(x)} \# build pseudo-observations
\]

## graphically check if the data comes from a meta-Gumbel model
## Construction of Independence Copula Class Objects

### Description
Con structs an independence copula class object with its corresponding dimension.

### Usage
```r
indepCopula(dim = 2)
```

### Arguments
- `dim` : the dimension of the copula.
Value

An independence copula object of class "indepCopula".

See Also

archmCopula, ellipCopula, evCopula.

Examples

```r
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低bnd: 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")

### Description

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

### Usage

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

### Arguments

- `n` sample size when simulating the distribution of the test statistics under independence.
- `p` dimension of the data when simulating the distribution of the test statistics under independence.
- `m` maximum cardinality of the subsets of variables for which a test statistic is to be computed. It makes sense to consider $m \ll p$ especially when $p$ is large.
- `N` number of repetitions when simulating under independence.
- `print.every` is deprecated in favor of `verbose`.
- `verbose` a logical specifying if progress should be displayed via `txtProgressBar`.
- `x` data frame or data matrix containing realizations (one per line) of the random vector whose independence is to be tested.
object of class "indepTestDist" as returned by the function indeptestsimHI().
It can be regarded as the empirical distribution of the test statistics under independence.

alpha

significance level used in the computation of the critical values for the test statistics.

test

object of class "indepTest" as returned by indeptestHI().
pvalues

logical indicating whether the dependogram should be drew from test statistics or the corresponding p-values.

print

logical indicating whether details should be printed.

Details

The current (C code) implementation of indeptestsimHI() uses (RAM) memory of size $O(n^2 p)$, and time

$O(N n^2 p)$. This renders it unfeasible when $n$ is large.

See the references below for more details, especially Genest and Rémillard (2004).

Value

The function indeptestsimHI() returns an object of class "indepTestDist" whose attributes are: sample.size, data.dimension, max.card.subsets, number.repetitions, subsets (list of the subsets for which test statistics have been computed), subsets.binary (subsets in binary 'integer' notation), dist.statistics.independence (a N line matrix containing the values of the test statistics for each subset and each repetition) and dist.global.statistic.independence (a vector a length N containing the values of the global Cramér-von Mises test statistic for each repetition – see Genest et al (2007), p.175).

The function indeptestHI() 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:
print(test)

## Display the dependogram with the details:
dependogram(test, print=TRUE)
}# (not in CRAN checks)

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

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

```r
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 chooses 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 $\hat{\theta}$, the corresponding Kendall’s tau is $\tau_{\text{G}}(\hat{\theta})$ where $\tau_{\text{G}}(\theta) = (\theta - 1)/\theta$ 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.
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

```r
## Definition of the function:
initOpt

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

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

## Initial interval:
initOpt("Gumbel") # contains theta

## Initial values:
initOpt("Gumbel", interval=FALSE, u=U) # 1.3195
initOpt("Gumbel", interval=FALSE, u=U, method="tau.mean") # 1.2844
```
Construct Simple "interval" Object

Description

Easy construction of an object of class `interval`, using typical mathematical notation.

Usage

```r
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

```r
interval("[0, 1]"

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

## infinite:
interval("[0, Inf)"

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

## str() to look at internals:
str( interval("[1.2, 7]")
```
Description

The S4 class \texttt{"interval"} is a simple class for numeric intervals.
\texttt{"maybeInterval"} is a class union (see \texttt{setClassUnion}) of \texttt{"interval"} and \texttt{"NULL"}.

Objects from the Class

Objects can be created by calls of the form \texttt{new("interval", \ldots)}, but typically they are built via \texttt{interval()}. 

Slots

\texttt{.Data}: numeric vector of length two, specifying the interval ranges.
\texttt{open}: \texttt{logical} vector of length two, specifying if the interval is open or closed on the left and right, respectively.

Extends

Class \texttt{"interval"} extends \texttt{"numeric"}, from data part, and \texttt{"maybeInterval"}, directly.

Methods

\texttt{"%in\%" signature(x = "numeric", table = "interval")}: check if \texttt{x} is inside the interval, carefully differentiating open and closed intervals.
\texttt{format signature(x = "interval"): \ldots}
\texttt{show signature(object = "interval"): \ldots}
\texttt{Summary signature(x = "interval"): Group methods, notably range(), min(), etc.}

Note

There are more sophisticated interval classes, functions and methods, notably in package \texttt{Intervals}. We only use this as a simple interface in order to specify our copula functions consistently.

Author(s)

Martin Maechler

See Also

\texttt{interval} constructs \texttt{"interval"} objects conveniently.

Examples

\begin{verbatim}
  -1:2 %in% interval("(0, Inf)")

  ## 0 is *not* inside
\end{verbatim}
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, \( p\text{K}() \), \( q\text{K}() \), \( d\text{K}() \), and \( r\text{K}() \) provide the distribution function, quantile function, density, and random number generator, respectively, for the Kendall distribution of an Archimedean copula.

Usage

```r
Kn(u, x)
pK(u, cop, d, n.MC=0, log.p=FALSE)
qK(u, cop, d, n.MC=0,
    method=c("default", "simple", "sort", "discrete", "monoH.FC"),
    u.grid, ...)
dK(u, cop, d, n.MC=0, log.p=FALSE)
rK(n, 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 \( r\text{K}() \) only) a outer_nacopula.
- **d** dimension (not used when \( \text{cop} \) is an outer_nacopula).
- **n.MC** integer, if positive, a Monte Carlo approach is applied with sample size equal to \( n.\text{MC} \) to evaluate the generator derivatives involved; otherwise (\( n.\text{MC}=0 \)) 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 \( u.\text{grid}=0:128/128 \).
  "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 $\psi$,

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

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)))

## 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(K.n(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)

## plot empirical distribution function based on W
## and the corresponding theoretical Kendall distribution function
## (exact and via Monte Carlo)
plot(ecdf(W), col="blue", xlab=expression(italic(u)), ylab=expression(italic(F[n](italic(u)))) -
main = expression("Empirical"- F[n]( C(U) ) -
"and its Kendall distribution"- K(u)),
do.points=FALSE, asp=1)
abline(0,1, lty=2); abline(h=0:1, v=0:1, lty=3, col="gray")
lines(u, Ku.MC, col="red")
lines(u, Ku, col="black")
legend(.2,.6, expression(F[n],K(u), K[M](u))),
col=c("blue","red","black"), lty=1, bty="n",
xjust = 0.25, yjust = 0.5)

### testing qK
uexpr <- quote( 0:63/63 ); u <- eval(uexpr)
log1mexp

```r
d <- 10
kop <- onacopula("Gumbel", list(theta = 2, 1:d))

system.time(Ku1 <- qK(u, kop@copula, d=d, method="simple"))
system.time(Ku2 <- qK(u, kop@copula, d=d, method="sort"))
system.time(Ku3 <- qK(u, kop@copula, d=d, method="discrete",
                       u.grid=0:1e4/1e4))
system.time(Ku4 <- qK(u, kop@copula, d=d, method="mono.H.FC",
                       u.grid=0:5e2/5e2))

cols <- adjustcolor(c("gray80", "light blue", "royal blue", "purple3"), 0.6)
matplot(u, cbind(Ku1,Ku2,Ku3,Ku4), type="l", lwd=2*4:1, lty = 1:4, col= cols,
        xlab=substitute(u), ylab=expression(k_{MQ}), legend="topleft",
        col=cols, lwd=2*4:1, lty=1:4, bty="n", inset=.03,
        legend= paste0("method=",
                      sQuote(c("simple", "sort", "discrete", "mono.H.FC")))
```

\[
\log(1 - \exp(-a)) \quad \text{or} \quad \log(1 + \exp(x))
\]

**Description**

Compute \( f(a) = \log(1 - \exp(-a)) \), respectively \( g(x) = \log(1 + \exp(x)) \) quickly numerically accurately.

**Usage**

\[
\log1mexp(a, \text{cutoff} = \log(2))
\]

\[
\log1pexp(x, c0 = -37, c1 = 18, c2 = 33.3)
\]

**Arguments**

- \(a\) numeric vector of positive values
- \(x\) numeric vector
- \(\text{cutoff}\) positive number; \(\log(2)\) is “optimal”,
  but the exact value is unimportant, and anything in \([0.5, 1]\) is fine.
- \(c0, c1, c2\) cutoffs for \(\log1pexp\); see below.

**Value**

\[
f(a) = \log(1 - \exp(-a)) \quad \text{or} \quad \log1p(-\exp(-a)) = \log(-\expm1(-a))
\]

or

\[
g(x) = \log(1 + \exp(x)) \quad \text{or} \quad \log1p(\exp(x))
\]

computed accurately and quickly

**Author(s)**

Martin Maechler, May 2002; \(\log1pexp()\) in 2012.
References


Examples

```r
a <- 2*seq(-58,10, length = 256)
expr <- expression(
  log(1 - exp(-a)),
  log(-exp(a)),
  log1p(-exp(-a)),
  log1mexp(a))
names(expr) <- c("DEF", "expm1", "log1p", "F")
str(expr)
head(expr)
# expm1() works here
tail(expr)
# 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, expr, type = "l", log = "x", col=col, lwd=lwd)
  legend("topleft", expr, col=col, lwd=lwd, lty=1:4, bty="n")
  # expm1() & log1mexp() work here

  matplot(a, -fa, type = "l", log = "xy", col=col, lwd=9)
  legend("left", paste("-", expr), col=col, lwd=lwd, lty=1:4, bty="n")
  # log1p() & log1mexp() work here
par(op)

curve(log1pexp, -10, 10, asp=1)
abline(0,1, h=0,v=0, lty=3, col="gray")

## Cutoff c1 for log1pexp() -- not often "needed":
curve(log1p(exp(x)) - log1pexp(x), 16, 20, n=2049)
## need for *some* cutoff:
x <- seq(700, 720, by=2)
cbind(x, log1p(exp(x)), log1pexp(x))

## Cutoff c2 for log1pexp():
curve((x+exp(-x)) - x, -10, 10, n=1025)
curve((x+exp(-x)) - x, 33.1, 33.5, n=1025)
```

---

**loss**

**LOSS and ALAE Insurance Data**

**Description**

Indemnity payment and allocated loss adjustment expense from an insurance company.
Usage
data(loss)

Format
A data frame with 1500 observations of the following 4 variables:

- **loss**: a numeric vector of loss amount up to the **limit**.
- **alae**: a numeric vector of the corresponding allocated loss adjustment expense.
- **limit**: a numeric vector of limit (-99 means no limit).
- **censored**: 1 means censored (limit reached) and 0 otherwise.

References

Examples
data(loss)

---

**Description**
sinc(x) computes the *sinc function* \( s(x) = \sin(x)/x \) for \( x \neq 0 \) and \( s(0) = 1 \), such that \( s() \) is continuous, also at \( x = 0 \).

A..Z(x, a) computes Zolotarev’s function to the power \( 1-a \).

Usage
sinc(x)
A..Z(x, alpha, I.alpha = 1 - alpha)

Arguments
- **x**: numeric argument in \([0, \pi]\), typically a vector.
- **alpha**: parameter in \((0,1]\).
- **I.alpha**: must be = \( 1 - \text{alpha} \), maybe more accurately when alpha is very close to 1.

Details
For more details about Zolotarev’s function, see, for example, Devroye (2009).
Value

\[ A_{\alpha}(x, \alpha) = \tilde{A}_{\alpha}(x, \alpha), \]
\[ = \frac{\sin(\alpha x)^\alpha \sin((1-\alpha)x)^{1-\alpha}}{\sin(x)}, \quad x \in [0, \pi], \]

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 Rémillard (see `indepTest`) for random vectors. The main difference comes from the fact that critical values and p-values are obtained through the bootstrap/permutation methodology, since, here, test statistics are not distribution-free.

Usage

```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 \( \text{sum}(d) = \text{ncol}(x) \).
- **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).

**References**


**See Also**

```r
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]

nc <- normalCopula(0.3, dim=3)

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

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

## Run the test

# silence output
suppressWarnings(test <- multIndepTest(x, d))
test

## Display the dependogram

dependogram(test, print=TRUE)
```

---

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

**Description**

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

**Usage**

```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 lag.max+1 especially when
lag.max is large.
N  
number of bootstrap/permutation samples.
alpha  
significance level used in the computation of the critical values for the test statistics.
print.every  
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 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[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**

```r
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

\[
\text{paramMargins} = \text{list}(\text{list(mean} = 0, \text{sd} = 2), \text{list(rate} = 2))
\]

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.

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))
```
mvdc-class

Class "mvdc"

Description
Class representing multivariate distributions constructed using Sklar's theorem.

Objects from the Class
Objects are typically created by \texttt{mvdc()}, or can be created by calls of the form \texttt{new("mvdc", \ldots)}.

Slots
- \texttt{copula}: Object of class "\texttt{copula}"}, specifying the copula.
- \texttt{margins}: Object of class "\texttt{character}"}, specifying the marginal distributions.
- \texttt{paramMargins}: Object of class "\texttt{list}"}, whose each component is a list of named components, giving the parameter values of the marginal distributions. See \texttt{mvdc}.
- \texttt{marginsIdentical}: Object of class "\texttt{logical}"}, that, if \texttt{TRUE}, restricts the marginal distributions to be identical, default is \texttt{FALSE}.

Methods
- \texttt{contour} signature(\texttt{x = "mvdc"}): ...  
- \texttt{dim} signature(\texttt{x = "mvdc"}): the dimension of the distribution; this is the same as \texttt{dim(x@copula)}.
- \texttt{persp} signature(\texttt{x = "mvdc"}): ...
- \texttt{show} signature(\texttt{object = "mvdc"}): quite compactly display the content of the "mvdc" object.

See Also
- \texttt{mvdc}, also for examples; for fitting, \texttt{fitMvdc}. 

```r
op <- par(mfcol=c(1,2))
pp <- persp(mv.d, pMvdc, xlim = c(-5,5), ylim = c(0,2),
main = "pMvdc(mv.d)", 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
```
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 (\( 1000 \times \text{system.time(.)[1]} \)) where \( k \) is \( \text{length(taus)} \). 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 \( V_{01} \)s for a particular \( \theta_0 \) and all the admissible \( \theta_1 \)s.

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

## takes about 7 seconds:% so we rather test a much smaller set in R CMD check
## Not run: nacFrail.time(10000, "Gumbel", taus= c(0.85,(1:9)/10, 0.95))
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
```
nacPairthetas

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 $T$, where $T[j,k] = \theta$ of the bivariate Archimedean copula $C(U_j, U_k)$.

Author(s)

Martin Maechler

See Also

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

```r
## test with
options(width=97)

(mm <- nacModel("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 <- nacModel("Gumbel", d=15, pr.comp = 1/8, order="seq")
stopifnot(isSymmetric(PT <- nacPairthetas(m1)))
m1; PT

m100 <- nacModel("Gumbel", d=100, pr.comp = 1/16, order="seq")
system.time(PT <- nacPairthetas(m100))# how slow (non-optimal algorithm)?
##-- very fast, still!
stopifnot(isSymmetric(PT))
m100

## image(PT)# not ok -- want one color per theta
nt <- length(th0 <- unique(sort(PT[!is.na(PT)])))
th1 <- c(th1[1]/2, th0, 1.25*th0[nt])
ths <- (th1[-1]+th1[-(nt+2)])/2
image(log(PT), breaks = ths, col = heat.colors(nt))
## Nicer and easier:
require(Matrix)
image(as(log(PT),"Matrix"), main = "log( nacPairthetas( m100 ))",
     useAbs=FALSE, useRaster=TRUE, border=NA)
```

---

**nesdepth**

*Nesting Depth of a Nested Archimedean Copula ("nacopula")*

**Description**

Compute the nesting depth of a nested Archimedean copula which is the length of the longest branch in the tree representation of the copula, and hence at least one.

**Usage**

`nesdepth(x)`

**Arguments**

- `x` object of class "nacopula".
onacopula

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

Constructing (Outer) Nested Archimedean Copulas

Description

Constructing (outer) nested Archimedean copulas (class outer_nacopula) is most conveniently done via onacopula(), using a nested C(...) notation.

Slightly less conveniently, but with the option to pass a list structure, onacopulaL() can be used, typically from inside another function programmatically.

Usage

onacopula (family, nacStructure)
onacopulaL(family, nacList)
nac2list(x)

Arguments

family either a character string, the short or longer form of the Archimedean family name (for example, "Clayton" or simply "C"); see the acopula-families documentation, or an acopula family object.
nacStructure a “formula” of the form

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

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(...) 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().

- 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 onacopula().

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 onacopula():
C10. <- onacopula("J", list(theta, 1:10))
stopifnot(identical(C10, C10.),
          identical(nac2list(C10), list(theta, 1:10)))

## Construct a three-dimensional nested Gumbel copula with parameters
## such that Kendall's tau of the respective bivariate margins are 0.2
## and 0.5.
theta0 <- copGumbel iTau(.2)
theta1 <- copGumbel iTau(.5)
C3 <- onacopula("G", C(theta0, 1, C(theta1, c(2,3))))

## Equivalent construction with onacopula():
str(NAlis <- list(theta0, 1, list(list(theta1, c(2,3)))))
C3. <- onacopula("Gumbel", NAlis)
stopifnot(identical(C3, C3.))

## An exercise: assume you got the copula specs as character string:
na3spec <- "C(theta@0, 1, C(theta@1, c(2,3)))"
```
Description

Build a new Archimedean copula by applying the outer power transformation to a given Archimedean copula.

Usage

opower(copbase, thetabase)

Arguments

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

thetabase  the univariate parameter \( \theta \) 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),
  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
```
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

Arguments

param a parameter vector.
d dimension of the resulting matrix.
P 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))))
```

pairsRosenblatt

Plots for Graphical GOF Test via Pairwise Rosenblatt Transforms

Description

pairsColList() creates a list containing information about colors for a given matrix of (approximate aka “pseudo”) p-values. These colors are used in pairsRosenblatt() for visualizing a graphical goodness-of-fit test based on pairwise Rosenblatt transformed data.

Usage

```r
pairsRosenblatt(cu.u, pvalueMat=pwiTest(pwiseIndepTest(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), ...)
```

```r
pairsColList(P, pdiv = c(1e-04, 0.001, 0.01, 0.05, 0.1, 0.5),
    signif.P = 0.05, pminP = 1e-05, bucketCols = NULL,
    fgColMat = NULL, bgColMat = NULL, col = "B&W.contrast",
    BWcutoff = 170,
    bg.col = c("ETHCL", "zurich", "zurich.by.fog", "baby",
```
Arguments

-cu.u- \((n,d,d)\)-array of pairwise Rosenblatt-transformed observations as returned by `pairwiseCopCop()`

-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 \(g1\) and \(g2\) correctly; see the code for more details.

-g1- function from \([0,1]^n \rightarrow [0,1]^n\) applied to "x" for plotting in one panel.

-g2- function from \([0,1]^{n\times2} \rightarrow [0,1]^n\) 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).

-pminP- 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 pminP should be in (0, \text{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 \texttt{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 \texttt{bwCutoff}), passed to \texttt{.pairsCond()}. If \texttt{collist} is not specified, this color is used to construct the points’ color.

\texttt{bwCutoff} number in \((0, 255)\) for switching foreground color if col="B\&W.contrast".

\texttt{bg.col} color scheme for the background colors.

\texttt{bg.ncol.gap} number of colors left out as "gap" for color buckets below/above signif.\(P\) (to make significance/non-significance more visible).

\texttt{bg.col.bottom} vector of length 3 containing a HCL color specification. If \texttt{bg.col.bottom} is provided and \texttt{bucketCols} is not, \texttt{bg.col.bottom} is used as the color for the bucket of smallest \(p\)-values.

\texttt{bg.col.top} vector of length 3 containing a HCL color specification. If \texttt{bg.col.top} is provided and \texttt{bucketCols} is not, \texttt{bg.col.top} is used as the color for the bucket of largest \(p\)-values.

Details

Extra arguments of \texttt{pairsRosenblatt()} are passed to \texttt{.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.

\texttt{pairsColList()} is basically an auxiliary function to specify the colors used in the graphical goodness-of-fit test as conducted by \texttt{pairsRosenblatt()}. The latter is described in detail in Hofert and Mächler (2013). See also \texttt{demo(gof_graph)}.

Value

\texttt{pairsRosenblatt: invisibly, the result of \texttt{.pairsCond()}}.

\texttt{pairsColList: a named list with components}

- \texttt{fgColMat} matrix of foreground colors.
- \texttt{bgColMat} matrix of background colors (corresponding to \(P\)).
- \texttt{bucketCols} vector containing the colors corresponding to \texttt{pvalueBuckets} as described above.
- \texttt{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@iTau(tau)  # correct parameter value
coph0 <- onacopulal(family, list(th, 1:d))  # define H0 copula
U. <- pobs(rCopula(n, cop=coph0))

## create array of pairwise coph0-transformed data columns
cu.u <- pairwiseCcop(U., coph0)
```
Methods for Function 'persp' in Package 'copula'

Description

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

Usage

```
## S4 method for signature 'copula'
persp(x, fun,
   n = 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 x 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 x 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 x ny matrix, as passed to persp.default.
persp the 4 x 4 transformation matrix returned by persp.default.

Methods

Perspective plots for both "copula" or "mvdc" objects, see x in the Arguments section.

See Also

The contour-methods for drawing contour lines of the same functions.

Examples

persp(frankCopula(-0.8), dCopula)
persp(claytonCopula(2), pCopula, main = "CDF of claytonCopula(2)"
## example with negative tau :
(th1 <- iTau(amhCopula(), -0.1))
persp(amhCopula(th1), dCopula)
persp(amhCopula(th1), 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

Description

Constructs a Plackett copula class object with its corresponding parameter.

Usage

plackettCopula(param)

Arguments

param a numeric vector specifying the parameter values.

Value

A Plackett copula object of class "plackettCopula".

References


See Also

ellipCopula, archmCopula.

Examples

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

pnacopula

Evaluation of (Nested) Archimedean Copulas

Description

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

Usage

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

## *Deprecated*:
pnacopula(x, u)
### Arguments

copula, x  
(nested) Archimedean copula of dimension $d$, that is, an object of class nacopula, typically from onacopula(\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

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

The value of a nested Archimedean copula is defined similarly. Note that a $d$-dimensional copula is called nested Archimedean if it is an Archimedean copula with arguments possibly replaced by other nested Archimedean copulas.

### Value

A numeric in $[0,1]$ which is the copula evaluated at $u$. (Currently not parallelized.)

### Note

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

### 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**

```
pobs(x, na.last = "keep",
     ties.method = c("average", "first", "random", "max", "min"),
     lower.tail = TRUE)
```

**Arguments**

- `x`  
  A `n × d`-matrix of random variates to be converted to pseudo-observations.
- `na.last`  
  boolean
  If `TRUE`, the last observation is kept for each column; otherwise, the first.
- `ties.method`  
  a character string
  Method used to deal with ties in the data and `rank` function; see `rank`.
- `lower.tail`  
  logical
  If `TRUE`, then the pseudo-observations are taken as 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 \( \text{pobs}(\cdot, \text{lower.tail} = \text{FALSE}) \) simply returns \( 1 - \text{pobs}(\cdot) \).

**Value**

A `n × d`-matrix of the same dimensions as `x` containing the pseudo-observations.

**Author(s)**

Marius Hofert

**Examples**

```
## Simple definition of the function:
pobs

## Draw from a multivariate normal distribution
d <- 10
set.seed(1)
P <- Matrix::nearPD(matrix(pmin(pmax(runif(d*d), 0.3), 0.99), ncol=d))$mat
```
polylog

polylog \(\text{Li}_s(z)\) 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

\[
\text{polylog}(z, s, \text{method} = \text{c("default", "sum", "negI-s-Stirling", "negI-s-Eulerian", "negI-s-asympt-w")}, \logarithm = \text{FALSE}, \text{is.log.z} = \text{FALSE}, \text{is.logmlog} = \text{FALSE}, \text{asymp.w.order} = 0, n.\text{sum})
\]

debye1(x)
debye2(x)
Arguments

- **z**: numeric or complex vector
- **s**: complex number; current implementation is aimed at $s \in \{0, -1, \ldots\}$
- **method**: a string specifying the algorithm to be used.
- **logarithm**: logical specified to return $\log(L_i(.))$ instead of $L_i(.)$
- **is.log.z**: logical; if TRUE, the specified $z$ argument is really $w = \log(z)$; that is, we compute $L_i(s(\exp(w)))$, and we typically have $w < 0$, or equivalently, $z < 1$.
- **is.logmlog**: logical; if TRUE, the specified argument $z$ is $lw = \log(-w) = \log(-\log(z))$ (where as above, $w = \log(z)$).
- **asymp.w.order**: currently only default is implemented.
- **n.sum**: for method="sum" only: the number of terms used.
- **x**: numeric vector, may contain Inf, NA, and negative values.

Details


For integer values of the polylogarithm order, the following explicit expressions are obtained by repeated application of $z \frac{\partial}{\partial z}$ to $L_i(z)$:

$$L_i(1) = -\log(1-z), \quad L_i(0) = \frac{z}{1-z}, \quad L_i(-1) = \frac{z}{(1-z)^2}, \quad L_i(-2) = \frac{z(1+z)}{(1-z)^3},$$

$$L_i(-3) = \frac{z(1+4z+z^2)}{(1-z)^4}, \text{ etc.}$$

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:

$$L_i(n) = \left(z \frac{\partial}{\partial z}\right)^n \frac{z}{1-z} = \sum_{k=0}^{n} k! S(n+1,k+1) \left(\frac{z}{1-z}\right)^{k+1} \quad (n = 0, 1, 2, \ldots),$$

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

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

$$L_i(n) = \frac{1}{(1-z)^{n+1}} \sum_{k=0}^{n-1} \binom{n}{k} z^{n-k} = \frac{z \sum_{k=0}^{n-1} \binom{n}{k} z^k}{(1-z)^{n+1}}, \quad (n = 1, 2, 3, \ldots),$$

where $\binom{n}{k}$ are the Eulerian numbers; see also [Eulerian](http://en.wikipedia.org/wiki/Eulerian_number).

Value

numeric/complex vector as $z$, or $x$, respectively.

Author(s)

Martin Maechler
polylog

References


For Debye functions: Levin (1981) above, and

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 \ [1, Inf)
## so x -> 1 is a limit case:
polylog(z = 1, s = 2)
## in the limit, should be equal to pi^2 / 6

## Default method uses GSL's dilog():
rLi2 <- curve(polylog(x, 2), -5, 1, n = 1+ 6*64, col=2, lwd=2)
abline(c(0,1), h=0, v=0:1, lty=3, col="gray40")
## "sum" method gives the same for |z| < 1 and large number of terms:
ii <- which(abs(rLi2$x) < 1)
stopifnot(all.equal(rLi2$y[ii],
                  polylog(rLi2$x[ii], 2, "sum", n.sum = 1e5),
                  tolerance = 1e-15))

z1 <- c(0.95, 0.99, 0.995, 0.999, 0.9999)
L1 <- polylog(z1, s=-3,method="negI-s-Euler") # close to Inf
LL1 <- polylog(log(z1), s=-3,method="negI-s-Euler",is.log.z=TRUE)
LLL1 <- polylog(log(-log(z1)),s=-3,method="negI-s-Euler",is.logmlog=TRUE)
all.equal(L1, LL1)
all.equal(L1, LLL1)

p.Li <- function(s.set, from = -2.6, to = 1/4, ylim = c(-1, 0.5),
                 colors = c("orange","brown", palette()), n = 201, ...)
{
  s.set <- sort(s.set, decreasing = TRUE)
s <- s.set[1] # _ for auto-ylab
curve(polylog(x, s, method="negI-s-Stirling"), from, to,
col=colors[1], ylim=ylim, n=n, ...)
```
abline(h=0, v=0, col="gray")
for(is in seq_along(s.set)[-1])
  curve(polylog(x, s=s.set[is], method="negI-s-Stirling"),
       add=TRUE, col = colors[is], n=n)
  s <- rev(s.set)
legend("bottomright", paste("s =", s), col=colors[2-s], lty=1, bty="n")
}

## yellow is unbearable (on white):
palette(local({p <- palette(); p[p="yellow"] <- "goldenrod"; p}))

## Wikipedia page plot (+/-):
p.Li(1:-3, ylim= c(-.8, .6), colors = c(2:4,6:7))

## and a bit more:
p.Li(1:-5)

## For the range we need it:
ccol <- c(NA,NA, rep(palette(),10))
p.Li(1:-20, from=0, to=.99, colors=ccol, ylim = c(0, 10))
## log-y scale:
p.Li(1:-20, from=0, to=.99, colors=ccol, ylim = c(.01, 1e7),
     log = "y", yaxt = "n")
if(require(sfsmisc)) eaxis(2) else axis(2)

---

**polynEval**

*Evaluate Polynomials*

**Description**

Evaluate a univariate polynomial at x (typically a vector), that is, compute, for a given vector of coefficients `coef`, the polynomial `coef[1] + coef[2]*x + ... + coef[p+1]*x^p`.

**Usage**

`polynEval(coef, x)`

**Arguments**

- `coef` numeric vector. If a vector, x can be an array and the result matches x.
- `x` numeric vector or array.

**Details**

The stable Horner rule is used for evaluation.

Using the C code speeds up the already fast R code available in `polyn.eval()` in package `sfsmisc`. 
printNacopula

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 sfsmisc for ages.

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

Examples
polynevalHcHQLMRLQIL x ] MRZWI C Hx M QI~R
polynevalHcHPL RTL MUPL SUL MQPL QIL
\[ x \cdot matrix(0:5, 2,3) \] # 5 zeros!

Description
Print a compact overview of a nested Archimedean copula, that is, an object of class "nacopula". Calling printNacopula explicitly allows to customize the printing behavior. Otherwise, the show() method calls printNacopula with default arguments only.

Usage
printNacopula(x, labelKids=NA, deltaInd=, indentNstr="", digits=getOption("digits"),
width=getOption("width"), ...)

Arguments
x an R object of class nacopula.
labelKids logical specifying if child copulas should be labeled; If NA (as per default), on each level, children are labeled only if they are not only-child.
deltaInd by how much should each child be indented more than its parent? (non-negative integer). The default is three with labelKids being the default or TRUE, otherwise it is five (for labelKids=FALSE).
indentNstr a character string specifying the indentation, that is, the string that should be prepended on the first line of output, and determine the amount of blanks for the remaining lines.
digits, width number of significant digits, and desired print width; see print.default.
... potentially further arguments, passed to methods.
Value

invisibly, x.

Author(s)

Martin Maechler

Examples

```r
C8 <- onacopula("R", C(1.9, 1,
  list(K1 = C(5.7, c(2,5)),
    abc= C(5.0, c(3,4,6),
    list(L2 = C(11.5, 7:8)))))
C8 # -> printNacopula(C8)
printNacopula(C8, delta=10)
printNacopula(C8, labelKids=TRUE)
```

---

**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

```r
prob(x, 1, u)
```

Arguments

- `x`: copula of dimension, that is, an object of class `Copula`.
- `1, u`: $d$-dimensional, numeric, lower and upper hypercube boundaries, respectively, satisfying $0 \leq l_i \leq u_i \leq 1$, for $i \in 1, \ldots, d$.

Value

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

Author(s)

Marius Hofert, Martin Maechler

See Also

- `pCopula(.)`.  

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))))

## Compute the probability of a random vector distributed according to
## this copula to fall inside the cube with lower point l and upper
## point u.
l <- c(.7,.8,.6)
u <- c(1,1,1)
prob(C3, l, u)

## ditto for a bivariate normal copula with rho = 0.8:
prob(normalCopula(.8), c(.2,.4), c(.3,.6))
```

---

**qqplot2**  

### Q-Q Plot with Rugs and Pointwise Asymptotic Confidence Intervals

**Description**

A Q-Q plot (possibly) with rugs and pointwise approximate (via the Central Limit Theorem) two-sided \(1-\alpha\) confidence intervals.

**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=paste0("Pointwise asymptotic \(\%\) 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, crop=NULL, ...)
```

**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().

crop  crop command, can be one of:

- NULL  crop with a default command (suitable for Unix).
- character  a string containing a crop command.
- ""  do not crop.

...  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^2,3 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)
Sampling Exponentially Tilted Stable Distributions

Description

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

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

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 "M" (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 "M", 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 \(\tilde{S}(\alpha, 1, \ldots)\); see above.

Author(s)

Marius Hofert, Martin Maechler
References


See Also

`rstableQ` for sampling stable distributions.

Examples

```r
## Draw random variates from an exponentially tilted stable distribution
## with given alpha, V0, and h = 1
alpha <- .2
V0 <- rgamma(200, 1)
rETS <- retstable(alpha, V0)

## Distribution plot the random variates -- log-scaled
hist(log(rETS), prob=TRUE)
lines(density(log(rETS)), col=2)
rug (log(rETS))
```

---

**Description**

`rF01Frank`: Generate a vector of random variates $V_{01} \sim F_{01}$ with Laplace-Stieltjes transform

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

for the given realizations $V_0$ of Frank’s $F_0$ and the parameters $\theta_0, \theta_1 \in (0, \infty)$ such that $\theta_0 \leq \theta_1$. 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 $\text{rej} < V_0 \theta_0 (1 - e^{-\theta_0}) V_0^{-1}$ a rejection from $F_{01}$ of Joe is applied (see `rF01Joe`; the meaning of the parameter approx is explained below), otherwise a sum is sampled with a logarithmic envelope for each summand.

`rF01Joe`: Generate a vector of random variates $V_{01} \sim F_{01}$ with Laplace-Stieltjes transform

$$\psi_{01}(t; V_0) = (1 - (1 - \exp(-t))^\alpha) V_0.$$ 

for the given realizations $V_0$ of Joe’s $F_0$ and the parameter $\alpha \in (0, 1]$. This distribution appears on sampling nested Joe copulas. Here, $\alpha = \theta_0/\theta_1$, where $\theta_0, \theta_1 \in [1, \infty)$ such that $\theta_0 \leq \theta_1$. 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

```r
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`.

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)
```

```r
# Sample corresponding V01 ~ F01 for Frank and Joe and plot histogram
```
rFFrankJoe

Sampling Distribution $F$ for Frank and Joe

Description

Generate a vector of variates $V \sim F$ from the distribution function $F$ with Laplace-Stieltjes transform

$$(1 - (1 - \exp(-t))(1 - e^{-\theta_1}))^\alpha/(1 - e^{-\theta_0}),$$

for Frank, or

$$1 - (1 - \exp(-t))^\alpha,$$

for Joe, respectively, where $\theta_0$ and $\theta_1$ denote two parameters of Frank (that is, $\theta_0, \theta_1 \in (0, \infty)$) and Joe (that is, $\theta_0, \theta_1 \in [1, \infty)$) satisfying $\theta_0 \leq \theta_1$ and $\alpha = \theta_0/\theta_1$.

Usage

```r
rFFrank(n, theta0, theta1, rej)
rFJoe(n, alpha)
```

Arguments

- **n**: number of variates from $F$.
- **theta0**: parameter $\theta_0$.
- **theta1**: parameter $\theta_1$.
- **rej**: method switch for rFFrank: if theta0 > rej a rejection from Joe’s family (Sibuya distribution) is applied (otherwise, a logarithmic envelope is used).
- **alpha**: parameter $\alpha = \theta_0/\theta_1$ in $(0, 1]$ for rFJoe.

Details

rFFrank(n, theta0, theta1, rej) calls rF01Frank(rep(1,n), theta0, theta1, rej, 1) and rFJoe(n, alpha) calls rSibuya(n, alpha).

Value

numeric vector of random variates $V$ of length $n$. 

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
\[ \texttt{rlog(n, p, Ip = 1 - p)} \]

Arguments
\begin{itemize}
  \item \texttt{n} sample size, that is, length of the resulting vector of random variates.
  \item \texttt{p} parameter in \((0, 1)\).
  \item \texttt{Ip} = 1 – \( p \), possibly more accurate, e.g., when \( p \approx 1 \).
\end{itemize}

Details
For documentation and didactical purposes, \texttt{rlogR} is a pure-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 integers of length \( n \) containing the generated random variates.

Author(s)
Marius Hofert, Martin Maechler
rnacModel

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 <- rnacModel("Gumbel", d=sample(20:25, 1), pr.comp = 0.3,
                   rtau0 = function() 0.25)
  m2 <- rnacModel("Joe", d=3, pr.comp = 0.1, order="each")
  mC <- rnacModel("Clayton", d=20, pr.comp = 0.3,
                  rtau0 = function() runif(1, 0.1, 0.5))
  mF <- rnacModel("Frank", d=sample(20:25, 1), pr.comp = 0.3, order="seq")
}
```

---

```r
desc
```

**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` (\(= \text{dim}(x)\)).

**Usage**

```r
rnacopula(n, copula, x, ...)
```

**Arguments**

- `n`: integer specifying the sample size, that is, the number of copula-distributed random vectors \(U_i\), to be generated.
- `copula`: an R object of class "outer_nacopula", typically from `onacopula()`.
- `x`: only for back compatibility: former name of `copula` argument.
- `...`: possibly further arguments for the given copula family.

**Details**

The generation happens by calling `rnchild()` on each child copula (which itself recursively descends the tree implied by the nested Archimedean structure). The algorithm is based on a mixture representation of the generic distribution functions \(F_0\) and \(F_{01}\) and is presented in McNeil (2008) and Hofert (2011a). Details about how to efficiently sample the distribution functions \(F_0\) and \(F_{01}\) can be found in Hofert (2010), Hofert (2012), and Hofert and Mächler (2011).

**Value**

- `numeric` matrix containing the generated vectors of random variates from the nested Archimedean copula object `copula`. 
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

\texttt{rnchild(x, theta0, V0, \ldots)}

Arguments

- \texttt{x} an "nacopula" object, typically emerging from an "outer_nacopula" object constructed with \texttt{onacopula()}.
- \texttt{theta0} the parameter (vector) of the parent Archimedean copula which contains \texttt{x} as a child.
- \texttt{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 \texttt{x} is generated.
- \ldots 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 \texttt{rnacopula()}.

Value

a list with components

- \texttt{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.
- \texttt{indcol} an integer vector of indices of \texttt{U} (the vector following a nested Archimedean copula of which \texttt{x} is a child) whose corresponding components of \texttt{U} are arguments of the nested Archimedean copula \texttt{x}.

Author(s)

Marius Hofert, Martin Maechler

See Also

\texttt{rnacopula}, also for the references. Further, classes "nacopula" and "outer_nacopula"; see also \texttt{onacopula}().

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.
theta0 <- copClayton@iTau(.2)
theta1 <- copClayton@iTau(.5)
C3 <- onacopula("C", C(theta0, 1, C(theta1, c(2,3))))
```
RSpobs

Pseudo-Observations of Radial and Uniform Part of Elliptical and Archimedean Copulas

Description

Given a matrix of iid multivariate data from a meta-elliptical or meta-Archimedean model, RSpobs() computes pseudo-observations of the radial part \( R \) and the vector \( S \) which follows a uniform distribution on the unit sphere (for elliptical copulas) or the unit simplex (for Archimedean copulas). These quantities can be used for (graphical) goodness-of-fit tests, for example.

Usage

RSpobs(x, do.pobs = TRUE, method = c("ellip", "archm"), ...)

Arguments

- **x**
  - an \((n, d)\)-matrix of data; if \( do.pobs=FALSE \), the rows of \( x \) are assumed to lie in the \( d \)-dimensional unit hypercube (if they do not, this leads to an error).

- **do.pobs**
  - logical indicating whether \( pobs() \) is applied to \( x \) for transforming the data to the \( d \)-dimensional unit hypercube.

- **method**
  - character string indicating the assumed underlying model, being meta-elliptical if \( method="ellip" \) (in which case \( S \) should be approximately uniform on the \( d \)-dimensional unit sphere) or meta-Archimedean if \( method="archm" \) (in which case \( S \) should be approximately uniform on the \( d \)-dimensional unit simplex).

- **...**
  - additional arguments passed to the implemented methods. These can be
    - method="ellip" \( \text{qG}(\cdot) \) (the quantile function of the (assumed) distribution function \( G_g \) as given in Genest, Hofert, G. Nešlehová (2014)); if provided, \( \text{qG}(\cdot) \) is used in the transformation for obtaining pseudo-observations of \( R \) and \( S \) (see the code for more details).
    - method="archm" \( \text{iPsi}(\cdot) \) (the assumed underlying generator inverse); if provided, \( \text{iPsi}(\cdot) \) 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 \textit{n-vector} containing the pseudo-observations of the radial part) and S (an \textit{(n,d)-matrix} containing the pseudo-observations of the uniform distribution (on the unit sphere/simplex)).

Author(s)

Marius Hofert

References


See Also

\texttt{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*: mvtnorm::rmvt(n, mean=mu, sigma=Sigma, df=nu)

## compute pseudo-observations of the radial part and uniform distribution
## once for 1a), once for 1b) below
RS.t <- RSpobs(X, method="ellip", qQ=qfunction(p) 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))

## 1) Graphically testing the radial part

## 1a) Q-Q plot of R against the correct quantiles
qqplot2(RS.t$R, qQ=function(p) sqrt(d*qt(p, df1=d, df2=nu))),
main.args=list(text=as.expression(substitute(bold(italic(
Random numbers from (Skew) Stable Distributions

```r
F[list(d, nu)](r^2/d) ~~ "Q-Q Plot",
list(d=d, nu=nu)),
side=3, cex=1.3, line=1.1, xpd=NA)

# 1b) Q-Q plot of \( R \) against the quantiles of \( F_R \) for a multivariate normal distribution

tit <- as.expression(substitute(bold(italic(chi[d])) ~~ "Q-Q Plot"), list(d=d))
qqplot2(RS.norm$R, qf=function(p) sqrt(qchisq(p, df=d)),
main.args=list(text=tit, side=3, cex=1.3, line=1.1, xpd=NA))

# 2) Graphically testing the angular distribution

# auxiliary function

qqp <- function(k, Bmat) {
  tit <- as.expression(substitute(plain("Beta")\(k/2\), \(nrow(Bmat)+1-k)/2\)))
  qqqplot2(Bmat[,k],
    df=function(p) qbeta(p, shape1=k/2, shape2=(ncol(Bmat)+1-k)/2),
    main.args=list(text=tit, side=3, cex=1.3, line=1.1, xpd=NA))
}

# 2a) Q-Q plot of the 'correct' angular distribution

(Bmat[,k] should follow a Beta(\(k/2, (d-k)/2\)) distribution)

Bmat.t <- gofBTstat(RS.t$S)
qqp(1, Bmat=Bmat.t) # k=1
qqp(3, Bmat=Bmat.t) # k=3

# 2b) Q-Q plot of the 'wrong' angular distribution

(Bmat.norm <- gofBTstat(RS.norm$S)
qqp(1, Bmat=Bmat.norm) # k=1
qqp(3, Bmat=Bmat.norm) # k=3

# 3) Graphically check independence between radial part and \( B_1 \) and \( B_3 \)

# 'correct' distributions (multivariate t)

plot(pobs(cbind(RS.t$R, Bmat.t[,1])), # k = 1
  xlab=quote(italic(R)), ylab=quote(italic(B)[1]),
  main=quote(bold("Rank plot between"~~ italic(R)~~ "and"~~ italic(B)[1])))

plot(pobs(cbind(RS.t$R, Bmat.t[,3])), # k = 3
  xlab=quote(italic(R)), ylab=quote(italic(B)[3]),
  main=quote(bold("Rank plot between"~~ italic(R)~~ "and"~~ italic(B)[3])))

# 'wrong' distributions (multivariate normal)

plot(pobs(cbind(RS.norm$R, Bmat.norm[,1])), # k = 1
  xlab=quote(italic(R)), ylab=quote(italic(B)[1]),
  main=quote(bold("Rank plot between"~~ italic(R)~~ "and"~~ italic(B)[1])))

plot(pobs(cbind(RS.norm$R, Bmat.norm[,3])), # k = 3
  xlab=quote(italic(R)), ylab=quote(italic(B)[3]),
  main=quote(bold("Rank plot between"~~ italic(R)~~ "and"~~ italic(B)[3])))
```
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

\[ \texttt{rstable1(n, alpha, beta, gamma = 1, delta = 0, pm = 1)} \]

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 \( \delta \in \mathbb{R} \).
- \( \text{pm} \) 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 \texttt{rstable1} provides two basic parametrizations, by default, \( \text{pm} = 1 \), 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).

\( \text{pm} = 0 \) 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 \texttt{rstable} (now in package \texttt{stabledist}) for Rmetrics; Martin Maechler vectorized it (also in \texttt{alpha},...), fixed it for \( \alpha = 1, \beta \neq 0 \) and sped it up.

References


safeUroot

One-dimensional Root (Zero) Finding - Extra "Safety" for Convenience

Description

safeUroot() as a “safe” version of uniroot() searches for a root (that is, zero) of the function \( f \) with respect to its first argument.

“Safe” means searching for the correct interval \( c(\text{lower}, \text{upper}) \) if \( \text{sign}(f(x)) \) 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 lower or upper endpoint, respectively.
- **Sig** desired sign of \( f(\text{upper}) \), or \textbf{NULL}.
- **check.conv** logical indicating whether a convergence warning of the underlying uniroot should be caught as an error.

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="red", lwd = 2)
```

See Also

\texttt{rstable} which also allows the 2-parametrization and provides further functionality for stable distributions.
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 \( S \cdot f(l) < 0 \) and \( S \cdot f(u) > 0 \).

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

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

Value

A list with four components, root, f.root, iter and estim.prec; see unroot.

Author(s)

Martin Maechler (from Martin’s R package nor1mix).

See Also

unroot.

Examples

```r
f1 <- function(x) (121 - x^2)/(x^2+1)
f2 <- function(x) exp(-x)*(x - 12)

try(unroot(f1, c(0,10)))
try(unroot(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) )
```
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 $m \ll lag.max+1$ especially when `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,\ldots,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.
```
show-methods

Methods for 'show' in Package 'copula'

Description

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

Methods

- `object = "copula"` see `Copula`.
- `object = "fitMvdc"` (see `fitMvdc`): and
- `object = "fitCopula"` (see `fitCopula`): these call the (hidden) `print` method, with its default argument. Using `print()` instead, allows to set digits, e.g.
- `object = "fitMvdc"` see `fitCopula`.
Description

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

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

its distribution function

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

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

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

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

$p\text{Sibuya}$ evaluates the distribution function.  
$d\text{Sibuya}$ evaluates the probability mass function.  
$r\text{Sibuya}$ generates random variates from $\text{Sib}(\alpha)$ with the algorithm described in Hofert (2011), Proposition 3.2.  
$d\text{sumSibuya}$ 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 $X_i \sim \text{Sib}(\alpha)$.  
This probability mass function can be shown (see Hofert (2010, pp. 99)) to be

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

Usage

$r\text{Sibuya}(n, \alpha)$  
$d\text{Sibuya}(x, \alpha, \log=FALSE)$  
$p\text{Sibuya}(x, \alpha, \text{lower.tail=}\text{TRUE, log.p=}\text{FALSE})$  
$d\text{sumSibuya}(x, n, \alpha, \text{method=}\text{c(}"\text{log", "direct", "diff", "exp.log", "Rmpfr", "Rmpfr0", "RmpfrM", "Rmpfr0M"}, \text{mpfr.ctrl} = \text{list(minPrec} = 21, \text{fac} = 1.25, \text{verbose=}\text{TRUE}), \text{log=}\text{FALSE})$

Arguments

$n$ for $r\text{Sibuya}$: sample size, that is, length of the resulting vector of random variates.  
for $d\text{sumSibuya}$: the number $n$ of summands. 
\alpha parameter in $(0, 1]$.  
\text{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 dsumSibuya 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 \( x \).
dsumSibuya: 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
SMI.12

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

```r
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", "SCMN.VX", "SYNN.VX", "SYST.VX", "RIGN.VX",
    "UBSN.VX", "ZURN.VX")
  lSMI <- sapply(symSMI, function(sym)
    get.hist.quote(instrument = sym, start= begSMI, end= endSMI,
      quote = "Close", provider = "yahoo",
      drop=TRUE))
  ## check if stock data have the same length for each company.
  sapply(lSMI, length)
  ## "concatenate" all:
  SMIo <- do.call(cbind, lSMI)
  ## and fill in the NAS:
  SMI.12 <- na.fill(SMIo, "extend")
  colnames(SMI.12) <- sub("\."VX", "", colnames(SMI.12))
  SMI.12 <- as.matrix(SMI.12)
}###----- original download

if(require(zoo)) {
  stopifnot(identical(SMI.12,
    local({ S <- as.matrix(na.fill(do.call(cbind, lSMI), "extend"))
       colnames(S) <- sub("\."VX", "", colnames(S)); S }))}

---

splom

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

**Details**

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

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

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

**Value**

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

Eulerian.all(n) is the same as `sapply(0:(n-1), Eulerian, n=n)` (for \(n > 0\)),
Stirling1.all(n) is the same as `sapply(1:n, Stirling1, n=n)`,
and
Stirling2.all(n) is the same as `sapply(1:n, Stirling2, n=n)`, but more efficient.
Note
For typical double precision arithmetic,
Eulerian(n, *) overflow (to Inf) for \( n \geq 172 \),
Stirling1(n, *) overflow (to ±Inf) for \( n \geq 171 \), and
Stirling2(n, *) overflow (to Inf) for \( n \geq 220 \).

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: http://dlmf.nist.gov/26.8

Examples
Stirling1(7,2)
Stirling2(7,3)
Stirling1all(9)
Stirling2all(9)

\begin{tabular}{l}
\textbf{tauAMH} & \textit{Ali-Mikhail-Haq ("AMH")’s and Joe’s Kendall’s Tau} \\
\end{tabular}

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

\texttt{tauAMH}(\texttt{theta})
\texttt{tauJoe}(\texttt{theta, method = c("hybrid", "digamma", "sum"), noTerms=446})

Arguments

\texttt{theta} \hspace{1cm} \text{numeric vector with values in } [-1, 1] \text{ for AMH, or } [0.238734, \text{Inf}] \text{ for Joe.}

\texttt{method} \hspace{1cm} \text{string specifying the method for } \texttt{tauJoe()}. \text{ Use the default, unless for research about the method. Up to } \texttt{copula} \text{ version 0.999-0, the only (implicit) method was "sum".}

\texttt{noTerms} \hspace{1cm} \text{the number of summation terms for the "sum" method; its default, 446 gives an absolute error smaller than } 10^{-5}.

Details

\texttt{tauAMH()}: \text{For small } \theta (= \theta), \text{ we use Taylor series approximations of up to order 7,}

\[
\tau_A(\theta) = \frac{2}{9} \theta \left(1 + \theta \left(\frac{1}{4} + \frac{\theta}{10} \left(1 + \theta \left(\frac{1}{2} + \theta \left(\frac{2}{7}\right)\right)\right)\right)\right) + O(\theta^6),
\]

where we found that dropping the last two terms (e.g., only using 5 terms from the } k = 7 \text{ term Taylor polynomial) is actually numerically advantageous.}

\texttt{tauJoe()}: \text{The "sum" method simply replaces the infinite sum by a finite sum (with } \texttt{noTerms} \text{ terms. The more accurate or faster methods, use analytical summation formulas, using the \texttt{digamma} aka } \psi \text{ function, see, e.g., } \texttt{http://en.wikipedia.org/wiki/Digamma_function#Series_formula}.\text{ The smallest sensible } \theta \text{ value, i.e., th for which } \texttt{tauJoe(th) == -1} \text{ is easily determined via } \texttt{str(uniroot(function(th) \texttt{tauJoe(th)}(-1), c(0.1, 0.3), tol = 1e-17), digits=12)} \text{ to be} 0.2387339899.\text{ }

Value

\text{a vector of the same length as } \texttt{theta (=} \theta), \text{ with } \tau \text{ values}

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

for \texttt{tauJoe}: \text{in [-1,1].}

Author(s)

Martin Maechler

See Also

\texttt{acopula-families}, and their class definition, "\texttt{acopula}".
Uranium Exploration Dataset of Cook & Johnson (1986)

Description

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

Usage

data(uranium)

Format

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

U (numeric) log concentration of Uranium.
Li (numeric) log concentration of Lithium.
Co (numeric) log concentration of Cobalt.
K (numeric) log concentration of Potassium.
Cs (numeric) log concentration of Cesium.
Sc (numeric) log concentration of Scandium.
Ti (numeric) log concentration of Titanium.

References


Examples

data(uranium)
Index

*Topic arith
  Bernoulli, 21
  interval, 96
  polylog, 129
  polyEval, 132
  Stirling, 162

*Topic array
  p2P, 119

*Topic classes
  acopula-class, 9
  archmCopula-class, 18
  copula-class, 35
  ellipCopula-class, 41
  evCopula-class, 53
  fgmCopula-class, 62
  fitCopula-class, 68
  indepCopula-class, 90
  interval-class, 97
  mvdc-class, 110
  nacopula-class, 112

*Topic datasets
  copFamilies, 30
  loss, 102
  rdj, 137
  SMI, 12, 159
  uranium, 165

*Topic distribution
  absdPsiMC, 8
  acR, 12
  archmCopula, 16
  betaBlomqvist, 22
  cCopula, 27
  Copula, 33
  dDiag, 36
  dnacopula, 38
  ellipCopula, 40
  evCopula, 51
  fgmCopula, 61
  gnacopula, 76
  gofOtherTstat, 83
  gofTstat, 84
  gtrafo, 86
  indepCopula, 89
  K, 98
  Mvdc, 108
  opower, 117
  plackettCopula, 126
  pnacopula, 126
  prob, 134
  retstable, 138
  rf01FrankJoe, 139
  rFFrankJoe, 141
  rlog, 142
  rnaacModel, 143
  rnacopula, 144
  rnchild, 145
  rstable1, 150
  Sibuya, 156
  tauAMH, 163

*Topic goodness-of-fit
  gofCopula, 78
  gofOtherTstat, 83
  gofTstat, 84

*Topic hplot
  .pairsCond, 6
  contour-methods, 28
  pairsRosenblatt, 120
  persp-methods, 124
  qqplot2, 135
  splom2, 160

*Topic htest
  An, 14
  evTestA, 54
  evTestC, 55
  evTestK, 57
  exchEVTest, 58
  exchTest, 59
  ggraph-tools, 74
gnacopula, 76
gofCopula, 78
gofEVCopula, 81
gofOtherTstat, 83
gofTstat, 84
indepTest, 91
multIndepTest, 104
multSerialIndepTest, 106
serialIndepTest, 153

*Topic manip
allComp, 13
getAcop, 73
p2P, 119
setTheta, 155

*Topic math
log1mexp, 101
mathFun, 103

*Topic methods
contour-methods, 28
persp-methods, 124
prob, 134
show-methods, 156

*Topic models
emde, 42
emle, 44
enacopula, 47
estim.misc, 49
fitCopula, 63
fitMvdc, 69
gofCopula, 78
gofEVCopula, 81

*Topic multivariate
.ac.classNames (getAcop), 73
.ac.longNames (getAcop), 73
.ac.objNames (getAcop), 73
.ac.shortNames (getAcop), 73
.emle, 47
.emle (emle), 44
.pairsCond, 6, 121, 122

%in%, numeric, interval-method
(interval-class), 97

A, 15
A (generator), 71
A, asymCopula-method (generator), 71
A, galambosCopula-method (generator), 71
INDEX

A, gumbelCopula-method (generator), 71
A, huslerReissCopula-method (generator), 71
A, indepCopula-method (generator), 71
A, tawnCopula-method (generator), 71
A, tevCopula-method (generator), 71
A-methods (generator), 71
A..Z (math-fun), 103
absdPsiMC, 8, 87
acopula, 5, 17, 20, 23, 30, 31, 34, 36, 37, 39, 73, 98, 111, 112, 115–118, 155, 164
acopula (acopula-class), 9
acopula-class, 9
acopula-families, 73
acopula-families (copFamilies), 30
acR, 12
Afun (generator), 71
AfunDer (generator), 71
callComp, 13
amhCopula (archmCopula), 16
amhCopula-class (archmCopula-class), 18
An, 14, 52, 55–57, 72, 83
Anfun (An), 14
archmCopula, 5, 16, 18, 19, 34–36, 41, 52, 53, 73, 90, 91, 109, 126
archmCopula-class, 18
array, 74, 75, 121
assocMeasures, 19
axis, 7
Bernoulli, 21
beta (beta.Blomqvist), 22
beta.Blomqvist, 22
beta.hat (beta.Blomqvist), 22
beta (beta.Blomqvist), 22
C.n, 24
cacopula (cCopula), 27
calibKendallsTau (assocMeasures), 19
calibSpearmansRho (assocMeasures), 19
cCopula, 27
character, 25, 37, 38, 43, 47, 64, 73, 75–78, 85, 94, 115, 121, 133, 135, 136, 147
class, 80, 82, 97, 144
claytonCopula (archmCopula), 16
claytonCopula-class (archmCopula-class), 18
Cn (C.n), 24
coef, 64, 68
copfamilies
cofittedMV (fitMvdc), 69
complex_dilog, 129
contour, 28, 29
contour, copula-method (contour-methods), 28
contour, indepCopula-method (contour-methods), 28
contour, mvdc-method (contour-methods), 28
contour-methods, 28
copAMH, 11
copAMH (copFamilies), 30
copClayton, 11, 17, 33
copClayton (copFamilies), 30
copFamilies, 30
copFrank, 11
copFrank (copFamilies), 30
copGumbel, 5, 11, 17, 20, 33, 116, 145
copGumbel (copFamilies), 30
copJoe, 11
copJoe (copFamilies), 30
copula, 33, 33, 61, 65, 70, 87, 134, 155, 156
copula, 18–20, 27, 29, 33, 34, 42, 53, 63, 68, 72, 78, 108–110, 125
Copula-class (copula-class), 35
copula-class, 35
copula-package, 4
cor, 50
dAdu (generator), 71
dAdu, galambosCopula-method (generator), 71
dAdu, gumbelCopula-method (generator), 71
dAdu, huslerReissCopula-method (generator), 71
dAdu, tawnCopula-method (generator), 71
dAdu, tevCopula-method (generator), 71
dAdu-methods (generator), 71
data.frame, 105
dCn (C.n), 24
dCopula, 5, 18, 29, 39, 125
dCopula (Copula), 33
dcoppula (Copula), 33
dCopula, matrix, archmCopula-method (Copula), 33
dCopula, matrix, asymCopula-method (Copula), 33
dCopula, matrix, asymExplicitCopula-method (Copula), 33
ebeta, 48, 95
ebta (estim.misc), 49
edmle, 47, 48, 95
edmle (estim.misc), 49
ellipCopula, 3, 17, 34–36, 40, 41, 42, 52, 65, 79, 87, 90, 109, 119, 120, 126
ellipCopula-class, 41
embedFonts, 38
eMde, 42, 47, 48, 50, 88, 95, 99
eMle, 44, 48, 50, 65, 95, 131
enacopula, 44–46, 47, 48, 50, 76, 95
environment, 118
estim.misc, 49
etau, 48
etau (estim.misc), 49
Eulerian, 5, 22, 130
Eulerian (Stirling), 162
evCopula, 5, 15, 17, 19, 35, 36, 51, 53, 55–57, 82, 83, 90, 91
evCopula-class, 53
evTestA, 15, 54, 56, 57, 83
evTestC, 15, 53, 55, 57, 83
evTestK, 15, 53, 55, 56, 57, 83
exchEVTest, 15, 58, 60
exchTest, 59, 59

F.n (C.n), 24
FALSE, 43
fgmCopula, 34–36, 61, 61
fgmCopula-class, 62
findInterval, 99
fitCopula, 5, 41, 61, 63, 64, 68–70, 79, 80, 156
fitCopula-class, 68
fitdist, 70
fitMvdc, 68, 69, 70, 110, 156
fitMvdc-class (fitCopula-class), 68
fittedMvdc-class (fitCopula-class), 68
formals, 108
format, interval-method (interval-class), 97
frankCopula (archmCopula), 16
frankCopula-class (archmCopula-class), 18
function, 10, 11, 29, 75, 121, 125, 143, 147
galambosCopula, 52, 53
galambosCopula (evCopula), 51
galambosCopula-class (evCopula-class), 53
generator, 71
genFun (generator), 71
genFunDer1 (generator), 71
genFunDer2 (generator), 71
genInv (generator), 71
getaC (phi), 11, 31, 73
getaSigma, 40, 41
ggetSigma (p2P), 119
gggraph-tools, 74
gnacopula, 76
gofBTstat (gofOtherTstat), 83
gofCopula, 44, 59, 60, 65, 70, 77, 78, 78, 83, 84, 86, 88
gofEVCopula, 15, 52, 53, 55–57, 81
gofMB (gofCopula), 78
gofOtherTstat, 83
gofPB (gofCopula), 78
gofTstat, 76, 78, 80, 84
gpvITest (gggraph-tools), 74
gTrafo, 78, 86
gumbelCopula, 52, 53
gumbelCopula (archmCopula), 16
gumbelCopula-class (archmCopula-class), 18
hTrafo, 43, 44, 76–79, 99
hTrafo (gTrafo), 86
huslerReissCopula, 52
huslerReissCopula (evCopula), 51
huslerReissCopula-class (evCopula-class), 53
image, 7
indepCopula, 16, 89, 90, 91
indepCopula-class, 90
indepTest, 74, 75, 91, 104, 105, 107, 153, 154
indepTestSim, 74, 75
indepTestSim (indepTest), 91
initialize.acopula-method (acopula-class), 9
initOpt, 94
integer, 14, 25, 45, 48, 75, 87, 98, 112, 140, 142, 146, 150, 157, 158
interval, 10, 11, 96, 96, 97
interval-class, 97
invisible, 38, 125, 136
iPsi (generator), 71
pairwiseCcop (ggraph-tools), 74
pairwiseIndepTest (ggraph-tools), 74
par, 7
pCopula, 5, 18, 26, 29, 125, 127, 134
pCopula (Copula), 33
pcopula (Copula), 33
pCopula, matrix, amhCopula-method (Copula), 33
pCopula, matrix, asymCopula-method (Copula), 33
pCopula, matrix, asymExplicitCopula-method (Copula), 33
pCopula, matrix, claytonCopula-method (Copula), 33
pCopula, matrix, fgmCopula-method (Copula), 33
pCopula, matrix, frankCopula-method (Copula), 33
pCopula, matrix, galambosCopula-method (Copula), 33
pCopula, matrix, gumbelCopula-method (Copula), 33
pCopula, matrix, huslerReissCopula-method (Copula), 33
pCopula, matrix, indepCopula-method (Copula), 33
pCopula, matrix, joeCopula-method (Copula), 33
pCopula, matrix, nacopula-method (pnacopula), 126
pCopula, matrix, normalCopula-method (Copula), 33
pCopula, matrix, plackettCopula-method (Copula), 33
pCopula, matrix, tawnCopula-method (Copula), 33
pCopula, matrix, tCopula-method (Copula), 33
pCopula, matrix, tevCopula-method (Copula), 33
pCopula, numeric, amhCopula-method (Copula), 33
pCopula, numeric, asymCopula-method (Copula), 33
pCopula, numeric, asymExplicitCopula-method (Copula), 33
pCopula, numeric, claytonCopula-method (Copula), 33
pCopula, numeric, fgmCopula-method
   (Copula), 33
pCopula, numeric, frankCopula-method
   (Copula), 33
pCopula, numeric, galambosCopula-method
   (Copula), 33
pCopula, numeric, gumbelCopula-method
   (Copula), 33
pCopula, numeric, huslerReissCopula-method
   (Copula), 33
pCopula, numeric, indepCopula-method
   (Copula), 33
pCopula, numeric, joeCopula-method
   (Copula), 33
pCopula, numeric, nacopula-method
   (pnacopula), 126
pCopula, numeric, normalCopula-method
   (Copula), 33
pCopula, numeric, plackettCopula-method
   (Copula), 33
pCopula, numeric, tawnCopula-method
   (Copula), 33
pCopula, numeric, tcopula-method
   (Copula), 33
pCopula, numeric, tevCopula-method
   (Copula), 33

pdf, 136
pdf.end, 38
persp, 124, 125
persp, copula-method (persp-methods), 124
persp, mvdc-method (persp-methods), 124
persp-methods, 124
persp.default, 125
pK (K), 98
plackettCopula, 126, 126
plackettCopula-class (copula-class), 35
plot, 136
pMvdc (Mvdc), 108
pmvdc (Mvdc), 108
pnacopula, 72, 126
pobs, 25, 26, 47, 63, 64, 69, 76, 87, 128, 147, 148
points, 121
polylog, 5, 129
polyEval, 132
predict.polynomial, 133
print, 156

print.default, 133
printNacopula, 112, 133
prob, 5, 134
prob, Copula-method (prob), 134
prob-methods (prob), 134
psi (generator), 71
psi, amhCopula-method (generator), 71
psi, claytonCopula-method (generator), 71
psi, frankCopula-method (generator), 71
psi, gumbelCopula-method (generator), 71
psi, joeCopula-method (generator), 71
psi-methods (generator), 71
pSibuya (Sibuya), 156
psiDabsMc (absdPsiMC), 8
psigma, 163
pvitTest (ggraph-tools), 74

qacr (acr), 12
qK, 87
qK (K), 98
qqline, 121, 136
qqplot2, 135

range, 29, 97, 125
rank, 128
rCopula, 5
rCopula (Copula), 33
rcopula (Copula), 33
rcopula, numeric, amhCopula-method
   (Copula), 33
rcopula, numeric, asymCopula-method
   (Copula), 33
rcopula, numeric, asymExplicitCopula-method
   (Copula), 33
rcopula, numeric, claytonCopula-method
   (Copula), 33
rcopula, numeric, evCopula-method
   (Copula), 33
rcopula, numeric, fgmCopula-method
   (Copula), 33
rcopula, numeric, frankCopula-method
   (Copula), 33
rcopula, numeric, galambosCopula-method
   (Copula), 33
rcopula, numeric, gumbelCopula-method
   (Copula), 33
rcopula, numeric, huslerReissCopula-method
   (Copula), 33
rCopula, numeric, indepCopula-method (Copula), 33
rCopula, numeric, joeCopula-method (Copula), 33
rCopula, numeric, nacopula-method (Copula), 33
rCopula, numeric, normalCopula-method (Copula), 33
rCopula, numeric, plackettCopula-method (Copula), 33
rCopula, numeric, tCopula-method (Copula), 33
rdj, 137
retstable, 104, 138
retstableR (retstable), 138
rF01Frank, 141, 142
rF01Frank (rF01FrankJoe), 139
rF01Joe, 142, 159
rF01Joe (rF01FrankJoe), 139
rFFrank, 140
rFFrank (rFFrankJoe), 141
rFFrankJoe, 141
rFJoe, 140, 159
rFJoe (rFFrankJoe), 141
rho, 5, 131
rho (assocMeasures), 19
rho, acopula-method (assocMeasures), 19
rho, amhCopula-method (assocMeasures), 19
rho, ANY-method (assocMeasures), 19
rho, claytonCopula-method (assocMeasures), 19
rho, copula-method (assocMeasures), 19
rho, evCopula-method (assocMeasures), 19
rho, fgmCopula-method (assocMeasures), 19
rho, frankCopula-method (assocMeasures), 19
rho, galambosCopula-method (assocMeasures), 19
rho, gumbelCopula-method (assocMeasures), 19
rho, huslerReissCopula-method (assocMeasures), 19
rho, indepCopula-method (assocMeasures), 19
rho, nacopula-method (assocMeasures), 19
rho, normalCopula-method (assocMeasures), 19
rho, plackettCopula-method (assocMeasures), 19
rho, tawncopula-method (assocMeasures), 19
rho, tCopula-method (assocMeasures), 19
rho, tevcopula-method (assocMeasures), 19
rho-methods (assocMeasures), 19
rK (K), 98
rlog, 142
rlogR (rlog), 142
rMvdc (Mvdc), 108
rmvdc (Mvdc), 108
rnacModel, 143, 145
rnacopula, 5, 140, 142, 143, 144, 146
rnchild, 144, 145, 145
rownames, 159
rSibuya, 140–142
rSibuya (Sibuya), 156
rSibuyaR (Sibuya), 156
RSpobs, 147
rstable, 150, 151
rstable (rstable1), 150
rstable1, 139, 149
rtrafo, 27, 28, 76–79
rtrafo (gtrafo), 86
rug, 136
safeUroot, 50, 151
serialIndepTest, 93, 105–107, 153
serialIndepTestSim (serialIndepTest), 153
setClassUnion, 35, 97
setTheta, 155
setTheta, acopula, ANY-method (setTheta), 155
setTheta, copula, ANY-method (setTheta), 155
setTheta, ellipCopula, ANY-method (setTheta), 155
setTheta, outer_nacopula, numeric-method (setTheta), 155
show, 133, 156
show, acopula-method (acopula-class), 9
show, copula-method (show-methods), 156
show, fitCopula-method (show-methods), 156
show, fitMvdc-method (show-methods), 156
show, interval-method (interval-class), 97
show,mvdc-method (mvdc-class), 110
show,nacopula-method (printNacopula), 133
show,normalCopula-method (show-methods), 156
show,tCopula-method (show-methods), 156
show-methods, 156
Sibuya, 5, 156
sinc (mathFun), 103
SMI.12, 159
spearmansRho (assocMeasures), 19
splom, 161
splom2, 160
Stirling, 162
Stirling1, 22
Stirling1 (Stirling), 162
Stirling2 (Stirling), 162
summary, 64, 70
summary, fitCopula-method (fitCopula-class), 68
summary, fitMvdc-method (fitCopula-class), 68
Summary, interval-method (interval-class), 97
summaryFitCopula-class (fitCopula-class), 68
summaryFitMvdc-class (fitCopula-class), 68
system.time, 111
tailIndex (assocMeasures), 19
tailIndex, acopula-method (assocMeasures), 19
tailIndex, amhCopula-method (assocMeasures), 19
tailIndex, ANY-method (assocMeasures), 19
tailIndex, claytonCopula-method (assocMeasures), 19
tailIndex, copula-method (assocMeasures), 19
tailIndex, evCopula-method (assocMeasures), 19
tailIndex, frankCopula-method (assocMeasures), 19
tailIndex, gumbelCopula-method (assocMeasures), 19
tailIndex, indepCopula-method (assocMeasures), 19
tau, 5, 17, 131
tau (assocMeasures), 19
tau, acopula-method (assocMeasures), 19
tau, amhCopula-method (assocMeasures), 19
tau, ANY-method (assocMeasures), 19
tau, archmCopula-method (assocMeasures), 19
tau, claytonCopula-method (assocMeasures), 19
tau, copula-method (assocMeasures), 19
tau, evCopula-method (assocMeasures), 19
tau, fgmCopula-method (assocMeasures), 19
tau, frankCopula-method (assocMeasures), 19
tau, galambosCopula-method (assocMeasures), 19
tau, gumbelCopula-method (assocMeasures), 19
tau, huslerReissCopula-method (assocMeasures), 19
tau, indepCopula-method (assocMeasures), 19
tau, joeCopula-method (assocMeasures), 19
tau, nacopula-method (assocMeasures), 19
tau, normalCopula-method (assocMeasures), 19
tau, plackettCopula-method (assocMeasures), 19
tau, tawnCopula-method (assocMeasures), 19
tau, tCOPULA-method (assocMeasures), 19
tau, tevCopula-method (assocMeasures), 19
tau-methods (assocMeasures), 19
tauAMH, 163
tauJoe (tauAMH), 163
tawnCopula, 52
tawnCopula (evCopula), 51
tawnCopula-class (evCopula-class), 53
tCopula, 35, 40, 119, 120
tCopula (ellipCopula), 40
tCopula-class (ellipCopula-class), 41
tevCopula, 52
tevCopula (evCopula), 51
tevCopula-class (evCopula-class), 53
trellis.par.get, 161
txtProgressBar, 76, 79, 82, 91, 105, 107, 153

uniroot, 12, 50, 98, 99, 151, 152
uranium, 165

vcov, 64, 68
vcov.fittedMV (fitMvdc), 69
vector, 119, 121, 122, 148
vignettes, 5