Package ‘rvinecopulib’

July 23, 2025

Type Package

Title High Performance Algorithms for Vine Copula Modeling

Version 0.7.3.1.0

Description Provides an interface to 'vinecopulib', a C++ library for vine copula modeling. The 'rvinecopulib' package implements the core features of the popular 'VineCopula' package, in particular inference algorithms for both vine copula and bivariate copula models. Advantages over 'VineCopula' are a sleeker and more modern API, improved performances, especially in high dimensions, nonparametric and multi-parameter families, and the ability to model discrete variables. The 'rvinecopulib' package includes 'vinecopulib' as header-only C++ library (currently version 0.7.2). Thus users do not need to install 'vinecopulib' itself in order to use 'rvinecopulib'. Since their initial releases, 'vinecopulib' is licensed under the MIT License, and 'rvinecopulib' is licensed under the GNU GPL version 3.

License GPL-3 | file LICENSE

Encoding UTF-8

NeedsCompilation yes

Depends R (>= 3.0.2)

Imports assertthat, graphics, grDevices, kde1d (>= 1.1.0), lattice, Rcpp (>= 0.12.12), stats, utils

Suggests igraph, ggplot2, ggraph, testthat

LinkingTo BH, Rcpp, RcppEigen, RcppThread (>= 2.1.2), wdm (>= 0.2.6), kde1d (>= 1.1.0)

BugReports `https://github.com/vinecopulib/rvinecopulib/issues`

URL `https://vinecopulib.github.io/rvinecopulib/`

RoxygenNote 7.3.2

Author Thomas Nagler [aut, cre], Thibault Vatter [aut]

Maintainer Thomas Nagler `<info@vinecopulib.org>`

Repository CRAN

Date/Publication 2025-06-13 12:20:02 UTC

Contents

as.bicop . . . 2
as_rvine_structure . . . 3
bicop . . . 5
bicop_dist . . . 8
bicop_distributions . . . 9
bicop_predict_and_fitted . . . 11
emp_cdf . . . 12
getters . . . 13
mBICV . . . 15
pairs_copula_data . . . 16
par_to_ktau . . . 17
plot.bicop_dist . . . 17
plot.rvine_structure . . . 18
plot.vinecop_dist . . . 19
pseudo_obs . . . 21
rosenblatt . . . 22
rvinecopulib . . . 23
rvine_structure . . . 25
rvine_structure_sim . . . 28
truncate_model . . . 29
vine . . . 30
vinecop . . . 32
vinecop_dist . . . 36
vinecop_distributions . . . 38
vinecop_predict_and_fitted . . . 39
vine_distributions . . . 41
vine_predict_and_fitted . . . 42

Index **44**

`as.bicop` *Convert list to bicop object*

Description

Convert list to bicop object

Usage

```
as.bicop(object, check = TRUE)
```

Arguments

`object`	a list containing entries for `"family"`, `"rotation"`, `"parameters"`, and `"npars"`.
`check`	whether to check for validity of the family/parameter specification.

Value

A bicop object corresponding to the specification in `object`.

Examples

```
as.bicop(list(family = "gumbel", rotation = 90, parameters = 2, npars = 1))
```

---

as_rvine_structure *Coerce various kind of objects to R-vine structures and matrices*

Description

`as_rvine_structure` and `as_rvine_matrix` are new S3 generics allowing to coerce objects into R-vine structures and matrices (see `rvine_structure()` and `rvine_matrix()`).

Usage

```
as_rvine_structure(x, ...)

as_rvine_matrix(x, ...)

## S3 method for class 'rvine_structure'
as_rvine_structure(x, ..., validate = FALSE)

## S3 method for class 'rvine_structure'
as_rvine_matrix(x, ..., validate = FALSE)

## S3 method for class 'list'
as_rvine_structure(x, ..., is_natural_order = FALSE)

## S3 method for class 'list'
as_rvine_matrix(x, ..., is_natural_order = FALSE)

## S3 method for class 'rvine_matrix'
as_rvine_structure(x, ..., validate = FALSE)

## S3 method for class 'rvine_matrix'
as_rvine_matrix(x, ..., validate = FALSE)

## S3 method for class 'matrix'
as_rvine_structure(x, ..., validate = TRUE)

## S3 method for class 'matrix'
as_rvine_matrix(x, ..., validate = TRUE)
```

Arguments

`x`	An object of class `rvine_structure`, `rvine_matrix`, `matrix` or `list` that can be coerced into an R-vine structure or R-vine matrix (see *Details*).
`...`	Other arguments passed on to individual methods.
`validate`	When 'TRUE", verifies that the input is a valid rvine-structure (see *Details*). You may want to suppress this when you know that you already have a valid structure and you want to save some time, or to explicitly enable it if you have a structure that you want to re-check.
`is_natural_order`	A flag indicating whether the `struct_array` element of `x` is assumed to be provided in natural order already (a structure is in natural order if the anti-diagonal is 1, .., d from bottom left to top right).

Details

The coercion to `rvine_structure` and `rvine_matrix` can be applied to different kind of objects Currently, `rvine_structure`, `rvine_matrix`, `matrix` and `list` are supported.

For `as_rvine_structure`:

- `rvine_structure` : the main use case is to re-check an object via `validate = TRUE`.
- `rvine_matrix` and `matrix` : allow to coerce matrices into R-vine structures (see `rvine_structure()` for more details). The main difference between `rvine_matrix` and `matrix` is the nature of the validity checks.
- `list` : must contain named elements `order` and `struct_array` to be coerced into an R-vine structure (see `rvine_structure()` for more details).

For `as_rvine_matrix`:

- `rvine_structure` : allow to coerce an `rvine_structure` into an R-vine matrix (useful e.g. for printing).
- `rvine_matrix`: similar to `as_rvine_structure` for `rvine_structure`, the main use case is to re-check an object via `validate = TRUE`.
- `matrix` : allow to coerce matrices into R-vine matrices (mainly by checking that the matrix defines a valid R-vine, see `rvine_matrix()` for more details).
- `list` : must contain named elements `order` and `struct_array` to be coerced into an R-vine matrix (see `rvine_structure()` for more details).

Value

Either an object of class `rvine_structure` or of class `rvine_matrix` (see `rvine_structure()` or `rvine_matrix()`).

See Also

rvine_structure rvine_matrix

Examples

```
# R-vine structures can be constructed from the order vector and struct_array
rvine_structure(order = 1:4, struct_array = list(
  c(4, 4, 4),
  c(3, 3),
  2
))

# ... or a similar list can be coerced into an R-vine structure
as_rvine_structure(list(order = 1:4, struct_array = list(
  c(4, 4, 4),
  c(3, 3),
  2
)))

# similarly, standard matrices can be coerced into R-vine structures
mat <- matrix(c(4, 3, 2, 1, 4, 3, 2, 0, 4, 3, 0, 0, 4, 0, 0, 0), 4, 4)
as_rvine_structure(mat)

# or truncate and construct the structure
mat[3, 1] <- 0
as_rvine_structure(mat)

# throws an error
mat[3, 1] <- 5
try(as_rvine_structure(mat))
```

`bicop` *Fit and select bivariate copula models*

Description

Fit a bivariate copula model for continuous or discrete data. The family can be selected automatically from a vector of options.

Usage

```
bicop(
  data,
  var_types = c("c", "c"),
  family_set = "all",
  par_method = "mle",
  nonpar_method = "quadratic",
  mult = 1,
  selcrit = "aic",
  weights = numeric(),
  psi0 = 0.9,
  presel = TRUE,
  allow_rotations = TRUE,
  keep_data = FALSE,
  cores = 1
)
```

Arguments

`data`	a matrix or data.frame with at least two columns, containing the (pseudo-)observations for the two variables (copula data should have approximately uniform margins). More columns are required for discrete models, see *Details*.
`var_types`	variable types, a length 2 vector; e.g., `c("c", "c")` for both continuous (default), or `c("c", "d")` for first variable continuous and second discrete.
`family_set`	a character vector of families; see *Details* for additional options.
`par_method`	the estimation method for parametric models, either `"mle"` for maximum likelihood or `"itau"` for inversion of Kendall's tau (only available for one-parameter families and `"t"`.
`nonpar_method`	the estimation method for nonparametric models, either `"constant"` for the standard transformation estimator, or `"linear"`/`"quadratic"` for the local-likelihood approximations of order one/two.
`mult`	multiplier for the smoothing parameters of nonparametric families. Values larger than 1 make the estimate more smooth, values less than 1 less smooth.
`selcrit`	criterion for family selection, either `"loglik"`, `"aic"`, `"bic"`, `"mbic"`. For `vinecop()` there is the additional option `"mbicv"`.
`weights`	optional vector of weights for each observation.
`psi0`	see `mBICV()`.
`presel`	whether the family set should be thinned out according to symmetry characteristics of the data.
`allow_rotations`	whether to allow rotations of the copula.
`keep_data`	whether the data should be stored (necessary for using `fitted()`).
`cores`	number of cores to use; if more than 1, estimation for multiple families is done in parallel.

Details

If there are missing data (i.e., `NA` entries), incomplete observations are discarded before fitting the copula.

Discrete variables:

When at least one variable is discrete, more than two columns are required for `data`: the first $n \times 2$ block contains realizations of $F_{X_1}(x_1), F_{X_2}(x_2)$. The second $n \times 2$ block contains realizations of $F_{X_1}(x_1^-), F_{X_2}(x_2^-)$. The minus indicates a left-sided limit of the cdf. For, e.g., an integer-valued variable, it holds $F_{X_1}(x_1^-) = F_{X_1}(x_1 - 1)$. For continuous variables the left limit and the cdf itself coincide. Respective columns can be omitted in the second block.

Family collections:
The family_set argument accepts all families in bicop_dist() plus the following convenience definitions:

- "all" contains all the families,
- "parametric" contains the parametric families (all except "tll"),
- "nonparametric" contains the nonparametric families ("indep" and "tll")
- "onepar" contains the parametric families with a single parameter,

("gaussian", "clayton", "gumbel", "frank", and "joe"),

- "twopar" contains the parametric families with two parameters, ("t", "bb1", "bb6", "bb7", and "bb8"),
- "threepar" contains the paramtric families with three parameters, ("tawn"),
- "elliptical" contains the elliptical families,
- "archimedean" contains the archimedean families,
- "ev" contains the extreme-value families,
- "BB" contains the BB families,
- "itau" families for which estimation by Kendall's tau inversion is available ("indep","gaussian", "t","clayton", "gumbel", "frank", "joe").

Value

An object inheriting from classes bicop and bicop_dist . In addition to the entries contained in bicop_dist(), objects from the bicop class contain:

- data (optionally, if keep_data = TRUE was used), the dataset that was passed to bicop().
- controls, a list with the set of fit controls that was passed to bicop().
- loglik the log-likelihood.
- nobs, an integer with the number of observations that was used to fit the model.

See Also

bicop_dist(), plot.bicop(), contour.bicop(), dbicop(), pbicop(), hbicop(), rbicop()

Examples

```
## fitting a continuous model from simulated data
u <- rbicop(100, "clayton", 90, 3)
fit <- bicop(u, family_set = "par")
summary(fit)

## compare fit with true model
contour(fit)
contour(bicop_dist("clayton", 90, 3), col = 2, add = TRUE)

## fit a model from discrete data
x_disc <- qpois(u, 1)  # transform to Poisson margins
plot(x_disc)
udisc <- cbind(ppois(x_disc, 1), ppois(x_disc - 1, 1))
fit_disc <- bicop(udisc, var_types = c("d", "d"))
summary(fit_disc)
```

`bicop_dist` *Bivariate copula models*

Description

Create custom bivariate copula models by specifying the family, rotation, parameters, and variable types.

Usage

```
bicop_dist(
  family = "indep",
  rotation = 0,
  parameters = numeric(0),
  var_types = c("c", "c")
)
```

Arguments

`family`	the copula family, a string containing the family name (see *Details* for all possible families).
`rotation`	the rotation of the copula, one of `0`, `90`, `180`, `270`.
`parameters`	a vector or matrix of copula parameters.
`var_types`	variable types, a length 2 vector; e.g., `c("c", "c")` for both continuous (default), or `c("c", "d")` for first variable continuous and second discrete.

Details

Implemented families:

type	name	name in R
-	Independence	"indep"
Elliptical	Gaussian	"gaussian"
"	Student t	"t"
Archimedean	Clayton	"clayton"
"	Gumbel	"gumbel"
"	Frank	"frank"
"	Joe	"joe"
"	Clayton-Gumbel (BB1)	"bb1"
"	Joe-Gumbel (BB6)	"bb6"
"	Joe-Clayton (BB7)	"bb7"
"	Joe-Frank (BB8)	"bb8"
Extreme-value	Tawn	"tawn"
Nonparametric	Transformation kernel	"tll"

Value

An object of class `bicop_dist`, i.e., a list containing:

- `family`, a `character` indicating the copula family.
- `rotation`, an `integer` indicating the rotation (i.e., either 0, 90, 180, or 270).
- `parameters`, a `numeric` vector or matrix of parameters.
- `npars`, a `numeric` with the (effective) number of parameters.
- `var_types`, the variable types.

See Also

`bicop_dist()`, `plot.bicop()`, `contour.bicop()`, `dbicop()`, `pbicop()`, `hbicop()`, `rbicop()`

Examples

```
## Clayton 90° copula with parameter 3
cop <- bicop_dist("clayton", 90, 3)
cop
str(cop)

## visualization
plot(cop)
contour(cop)
plot(rbicop(200, cop))

## BB8 copula model for discrete data
cop_disc <- bicop_dist("bb8", 0, c(2, 0.5), var_types = c("d", "d"))
cop_disc
```

`bicop_distributions` *Bivariate copula distributions*

Description

Density, distribution function, random generation and h-functions (with their inverses) for the bivariate copula distribution.

Usage

```
dbicop(u, family, rotation, parameters, var_types = c("c", "c"))

pbicop(u, family, rotation, parameters, var_types = c("c", "c"))

rbicop(n, family, rotation, parameters, qrng = FALSE)

hbicop(
  u,
  cond_var,
  family,
  rotation,
  parameters,
  inverse = FALSE,
  var_types = c("c", "c")
)
```

Arguments

`u`	evaluation points, a matrix with at least two columns, see *Details*.
`family`	the copula family, a string containing the family name (see `bicop` for all possible families).
`rotation`	the rotation of the copula, one of `0`, `90`, `180`, `270`.
`parameters`	a vector or matrix of copula parameters.
`var_types`	variable types, a length 2 vector; e.g., `c("c", "c")` for both continuous (default), or `c("c", "d")` for first variable continuous and second discrete.
`n`	number of observations. If ‘length(n) > 1“, the length is taken to be the number required.
`qrng`	if `TRUE`, generates quasi-random numbers using the bivariate Generalized Halton sequence (default `qrng = FALSE`).
`cond_var`	either `1` or `2`; `cond_var = 1` conditions on the first variable, `cond_var = 2` on the second.
`inverse`	whether to compute the h-function or its inverse.

Details

See bicop for the various implemented copula families.

The copula density is defined as joint density divided by marginal densities, irrespective of variable types.

H-functions (`hbicop()`) are conditional distributions derived from a copula. If $C(u, v) = P(U \leq u, V \leq v)$ is a copula, then

$$h_1(u, v) = P(V \leq v | U = u) = \partial C(u, v)/\partial u,$$

$$h_2(u, v) = P(U \leq u | V = v) = \partial C(u, v)/\partial v.$$

In other words, the H-function number refers to the conditioning variable. When inverting H-functions, the inverse is then taken with respect to the other variable, that is `v` when `cond_var = 1` and `u` when `cond_var = 2`.

Discrete variables:

When at least one variable is discrete, more than two columns are required for `u`: the first $n \times 2$ block contains realizations of $F_{X_1}(x_1), F_{X_2}(x_2)$. The second $n \times 2$ block contains realizations of $F_{X_1}(x_1^-), F_{X_2}(x_2^-)$. The minus indicates a left-sided limit of the cdf. For, e.g., an integer-valued variable, it holds $F_{X_1}(x_1^-) = F_{X_1}(x_1 - 1)$. For continuous variables the left limit and the cdf itself coincide. Respective columns can be omitted in the second block.

Value

`dbicop()` gives the density, `pbicop()` gives the distribution function, `rbicop()` generates random deviates, and `hbicop()` gives the h-functions (and their inverses).

The length of the result is determined by `n` for `rbicop()`, and the number of rows in `u` for the other functions.

The numerical arguments other than `n` are recycled to the length of the result.

Note

The functions can optionally be used with a bicop_dist object in place of the `family` argument, e.g., `dbicop(c(0.1, 0.5), bicop_dist("indep"))` or `hbicop(c(0.1, 0.5), 2, bicop_dist("indep"))`.

See Also

`bicop_dist()`, `bicop()`

Examples

```
## evaluate the copula density
dbicop(c(0.1, 0.2), "clay", 90, 3)
dbicop(c(0.1, 0.2), bicop_dist("clay", 90, 3))

## evaluate the copula cdf
pbicop(c(0.1, 0.2), "clay", 90, 3)

## simulate data
plot(rbicop(500, "clay", 90, 3))

## h-functions
joe_cop <- bicop_dist("joe", 0, 3)
# h_1(0.1, 0.2)
hbicop(c(0.1, 0.2), 1, "bb8", 0, c(2, 0.5))
# h_2^{-1}(0.1, 0.2)
hbicop(c(0.1, 0.2), 2, joe_cop, inverse = TRUE)

## mixed discrete and continuous data
x <- cbind(rpois(10, 1), rnorm(10, 1))
u <- cbind(ppois(x[, 1], 1), pnorm(x[, 2]), ppois(x[, 1] - 1, 1))
pbicop(u, "clay", 90, 3, var_types = c("d", "c"))
```

`bicop_predict_and_fitted` *Predictions and fitted values for a bivariate copula model*

Description

Predictions of the density, distribution function, h-functions (with their inverses) for a bivariate copula model.

Usage

```
## S3 method for class 'bicop_dist'
predict(object, newdata, what = "pdf", ...)

## S3 method for class 'bicop'
fitted(object, what = "pdf", ...)
```

Arguments

`object`	a `bicop` object.
`newdata`	points where the fit shall be evaluated.
`what`	what to predict, one of `"pdf"`, `"cdf"`, `"hfunc1"`, `"hfunc2"`, `"hinv1"`, `"hinv2"`.
`...`	unused.

Details

`fitted()` can only be called if the model was fit with the `keep_data = TRUE` option.

Discrete variables:

When at least one variable is discrete, more than two columns are required for `newdata`: the first $n \times 2$ block contains realizations of $F_{X_1}(x_1), F_{X_2}(x_2)$. The second $n \times 2$ block contains realizations of $F_{X_1}(x_1^-), F_{X_2}(x_2^-)$. The minus indicates a left-sided limit of the cdf. For, e.g., an integer-valued variable, it holds $F_{X_1}(x_1^-) = F_{X_1}(x_1 - 1)$. For continuous variables the left limit and the cdf itself coincide. Respective columns can be omitted in the second block.

Value

`fitted()` and `logLik()` have return values similar to `dbicop()`, `pbicop()`, and `hbicop()`.

Examples

```
# Simulate and fit a bivariate copula model
u <- rbicop(500, "gauss", 0, 0.5)
fit <- bicop(u, family = "par", keep_data = TRUE)

# Predictions
all.equal(predict(fit, u, "hfunc1"), fitted(fit, "hfunc1"),
          check.environment = FALSE)
```

---

emp_cdf *Corrected Empirical CDF*

Description

The empirical CDF with tail correction, ensuring that its output is never 0 or 1.

Usage

```
emp_cdf(x)
```

Arguments

x	numeric vector of observations

Details

The corrected empirical CDF is defined as

$$F_n(x) = \frac{1}{n+1} \max\left\{1, \sum_{i=1}^{n} 1(X_i \leq x)\right\}$$

Value

A function with signature `function(x)` that returns $F_n(x)$.

Examples

```
# fit ECDF on simulated data
x <- rnorm(100)
cdf <- emp_cdf(x)

# output is bounded away from 0 and 1
cdf(-50)
cdf(50)
```

`getters` *Extracts components of* `bicop_dist` *and* `vinecop_dist` *objects*

Description

Extracts either the structure matrix (for `vinecop_dist` only), or pair-copulas, their parameters, Kendall's taus, or families (for `bicop_dist` and `vinecop_dist`).

Usage

```
get_structure(object)

get_pair_copula(object, tree = NA, edge = NA)

get_parameters(object, tree = NA, edge = NA)

get_ktau(object, tree = NA, edge = NA)

get_family(object, tree = NA, edge = NA)
```

```
get_all_pair_copulas(object, trees = NA)

get_all_parameters(object, trees = NA)

get_all_ktaus(object, trees = NA)

get_all_families(object, trees = NA)
```

Arguments

`object`	a `bicop_dist`, `vinecop_dist` or `vine_dist` object.
`tree`	tree index (not required if `object` is of class `bicop_dist`).
`edge`	edge index (not required if `object` is of class `bicop_dist`).
`trees`	the trees to extract from `object` (`trees = NA` extracts all trees).

Details

#' The get_structure method (for `vinecop_dist` or `vine_dist` objects only) extracts the structure (see rvine_structure for more details).

The get_matrix method (for `vinecop_dist` or `vine_dist` objects only) extracts the structure matrix (see rvine_structure for more details).

The other `get_xyz` methods for `vinecop_dist` or `vine_dist` objects return the entries corresponding to the pair-copula indexed by its `tree` and `edge`. When `object` is of class `bicop_dist`, `tree` and `edge` are not required.

- `get_pair_copula()` = the pair-copula itself (see bicop).
- `get_parameters()` = the parameters of the pair-copula (i.e., a `numeric` scalar, vector, or matrix).
- `get_family()` = a character for the family (see bicop for implemented families).
- `get_ktau()` = a `numeric` scalar with the pair-copula Kendall's tau.

The `get_all_xyz` methods (for `vinecop_dist` or `vine_dist` objects only) return lists of lists, with each element corresponding to a tree in `trees`, and then elements of the sublists correspond to edges. The returned lists have two additional attributes:

- `"d"` = the dimension of the model.
- `"trees"` = the extracted trees.

Value

The structure matrix, or pair-copulas, their parameters, Kendall's taus, or families.

Examples

```
# specify pair-copulas
bicop <- bicop_dist("bb1", 90, c(3, 2))
pcs <- list(
  list(bicop, bicop), # pair-copulas in first tree
  list(bicop) # pair-copulas in second tree
)

# specify R-vine matrix
mat <- matrix(c(1, 2, 3, 1, 2, 0, 1, 0, 0), 3, 3)

# set up vine copula model
vc <- vinecop_dist(pcs, mat)

# get the structure
get_structure(vc)
all(get_matrix(vc) == mat)

# get pair-copulas
get_pair_copula(vc, 1, 1)
get_all_pair_copulas(vc)
all.equal(get_all_pair_copulas(vc), pcs,
          check.attributes = FALSE, check.environment = FALSE)
```

`mBICV` *Modified vine copula Bayesian information criterion (mBICv)*

Description

Calculates the modified vine copula Bayesian information criterion.

Usage

```
mBICV(object, psi0 = 0.9, newdata = NULL)
```

Arguments

`object`	a fitted `vinecop` object.
`psi0`	baseline prior probability of a non-independence copula.
`newdata`	optional; a new data set.

Details

The modified vine copula Bayesian information criterion (mBICv) is defined as

$$BIC = -2loglik + \nu log(n) - 2\sum_{t=1}^{d-1}(q_t log(\psi_0^t) - (d-t-q_t)log(1-\psi_0^t))$$

where loglik is the log-likelihood and ν is the (effective) number of parameters of the model, t is the tree level ψ_0 is the prior probability of having a non-independence copula and q_t is the number of non-independence copulas in tree t. The mBICv is a consistent model selection criterion for parametric sparse vine copula models.

References

Nagler, T., Bumann, C., Czado, C. (2019). Model selection for sparse high-dimensional vine copulas with application to portfolio risk. *Journal of Multivariate Analysis, in press* (http://arxiv.org/pdf/1801.09739)

Examples

```
u <- sapply(1:5, function(i) runif(50))
fit <- vinecop(u, family = "par", keep_data = TRUE)
mBICV(fit, 0.9) # with a 0.9 prior probability of a non-independence copula
mBICV(fit, 0.1) # with a 0.1 prior probability of a non-independence copula
```

---

pairs_copula_data *Exploratory pairs plot for copula data*

Description

This function provides pair plots for copula data. It shows bivariate contour plots on the lower panel, scatter plots and correlations on the upper panel and histograms on the diagonal panel.

Usage

```
pairs_copula_data(data, main = "", ...)
```

Arguments

`data`	the data (must lie in the unit hypercube).
`main`	an overall title for the plot.
`...`	other parameters passed to `pairs.default()`, `contour.bicop()`, `points.default()`, `hist.default()`, or `bicop()`.

Examples

```
u <- replicate(3, runif(100))
pairs_copula_data(u)
```

`par_to_ktau` *Conversion between Kendall's tau and parameters*

Description

Conversion between Kendall's tau and parameters

Usage

```
par_to_ktau(family, rotation, parameters)

ktau_to_par(family, tau)
```

Arguments

`family`	a copula family (see `bicop_dist()`) or a bicop_dist object.
`rotation`	the rotation of the copula, one of `0`, `90`, `180`, `270`.
`parameters`	vector or matrix of copula parameters, not used when `family` is a `bicop_dist` object.
`tau`	Kendall's τ.

Examples

```
# the following are equivalent
par_to_ktau(bicop_dist("clayton", 0, 3))
par_to_ktau("clayton", 0, 3)

ktau_to_par("clayton", 0.5)
ktau_to_par(bicop_dist("clayton", 0, 3), 0.5)
```

`plot.bicop_dist` *Plotting tools for* `bicop_dist` *and* `bicop` *objects*

Description

There are several options for plotting bicop_dist objects. The density of a bivariate copula density can be visualized as surface/perspective or contour plot. Optionally, the density can be coupled with standard normal margins (default for contour plots).

Usage

```
## S3 method for class 'bicop_dist'
plot(x, type = "surface", margins, size, ...)

## S3 method for class 'bicop'
plot(x, type = "surface", margins, size, ...)

## S3 method for class 'bicop_dist'
contour(x, margins = "norm", size = 100L, ...)

## S3 method for class 'bicop'
contour(x, margins = "norm", size = 100L, ...)
```

Arguments

`x`	`bicop_dist` object.
`type`	plot type; either `"surface"` or `"contour"`.
`margins`	options are: `"unif"` for the original copula density, `"norm"` for the transformed density with standard normal margins, `"exp"` with standard exponential margins, and `"flexp"` with flipped exponential margins. Default is `"norm"` for `type = "contour"`, and `"unif"` for `type = "surface"`.
`size`	integer; the plot is based on values on a `size x size` grid, default is 100.
`...`	optional arguments passed to `graphics::contour()` or `lattice::wireframe()`.

See Also

`bicop_dist()`, `graphics::contour()`, `lattice::wireframe()`

Examples

```
## construct bicop_dist object for a student t copula
obj <- bicop_dist(family = "t", rotation = 0, parameters = c(0.7, 4))

## plots
plot(obj) # surface plot of copula density
contour(obj) # contour plot with standard normal margins
contour(obj, margins = "unif") # contour plot of copula density
```

`plot.rvine_structure` *Plotting R-vine structures*

Description

Plot one or all trees of an R-vine structure.

Usage

```
## S3 method for class 'rvine_structure'
plot(x, ...)

## S3 method for class 'rvine_matrix'
plot(x, ...)
```

Arguments

`x`	an `rvine_structure` or `rvine_matrix` object.
`...`	passed to `plot.vinecop_dist()`.

Examples

```
plot(cvine_structure(1:5))
plot(rvine_structure_sim(5))
mat <- rbind(c(1, 1, 1), c(2, 2, 0), c(3, 0, 0))
plot(rvine_matrix(mat))
plot(rvine_matrix_sim(5))
```

`plot.vinecop_dist` *Plotting* `vinecop_dist` *and* `vinecop` *objects.*

Description

There are two plotting generics for `vinecop_dist` objects. `plot.vinecop_dist` plots one or all trees of a given R-vine copula model. Edges can be labeled with information about the corresponding pair-copula. `contour.vinecop_dist` produces a matrix of contour plots (using `plot.bicop`).

Usage

```
## S3 method for class 'vinecop_dist'
plot(x, tree = 1, var_names = "ignore", edge_labels = NULL, ...)

## S3 method for class 'vinecop'
plot(x, tree = 1, var_names = "ignore", edge_labels = NULL, ...)

## S3 method for class 'vinecop_dist'
contour(x, tree = "ALL", cex.nums = 1, ...)

## S3 method for class 'vinecop'
contour(x, tree = "ALL", cex.nums = 1, ...)
```

Arguments

`x`	`vinecop_dist` object.
`tree`	`"ALL"` or integer vector; specifies which trees are plotted.
`var_names`	integer; specifies how to make use of variable names: • ‘"ignore"“ = variable names are ignored, • ‘"use"“ = variable names are used to annotate vertices, • ‘"legend"“ = uses numbers in plot and adds a legend for variable names, • ‘"hide"“ = no numbers or names, just the node.
`edge_labels`	character; options are: • `"family"` = pair-copula family (see `[bicop_dist()]`), • ‘"tau"“ = pair-copula Kendall’s tau • ‘"family_tau"“ = pair-copula family and Kendall’s tau, • ‘"pair"“ = the name of the involved variables.
`...`	Unused for `plot` and passed to `contour.bicop` for `contour`.
`cex.nums`	numeric; expansion factor for font of the numbers.

Details

If you want the contour boxes to be perfect squares, the plot height should be `1.25/length(tree)*(d - min(tree))` times the plot width.

The `plot()` method returns an object that (among other things) contains the `igraph` representation of the graph; see *Examples*.

Author(s)

Thomas Nagler, Thibault Vatter

See Also

`vinecop_dist`, `plot.bicop`

Examples

```
# set up vine copula model
u <- matrix(runif(20 * 10), 20, 10)
vc <- vinecop(u, family = "indep")

# plot
plot(vc, tree = c(1, 2))
plot(vc, edge_labels = "pair")

# extract igraph representation
plt <- plot(vc, edge_labels = "family_tau")
igr_obj <- get("g", plt$plot_env)[[1]]
igr_obj  # print object
igraph::E(igr_obj)$name  # extract edge labels
```

```
# set up another vine copula model
pcs <- lapply(1:3, function(j) # pair-copulas in tree j
  lapply(runif(4 - j), function(cor) bicop_dist("gaussian", 0, cor)))
mat <- rvine_matrix_sim(4)
vc <- vinecop_dist(pcs, mat)

# contour plot
contour(vc)
```

`pseudo_obs` *Pseudo-Observations*

Description

Compute the pseudo-observations for the given data matrix.

Usage

```
pseudo_obs(x, ties_method = "average", lower_tail = TRUE)
```

Arguments

`x`	vector or matrix random variates to be converted (column wise) to pseudo-observations.
`ties_method`	similar to `ties.method` of `rank()` (only `"average"`, `"first"` and `"random"` currently available).
`lower_tail`	`logical` which, if 'FALSE", returns the pseudo-observations when applying the empirical marginal survival functions.

Details

Given `n` realizations $x_i = (x_{i1}, \ldots, x_{id})$, $i \in \{1, \ldots, n\}$ 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.

When `lower_tail = FALSE`, then `pseudo_obs()` simply returns `1 - pseudo_obs()`.

Value

a vector of matrix of the same dimension as the input containing the pseudo-observations.

Examples

```
# pseudo-observations for a vector
pseudo_obs(rnorm(10))

# pseudo-observations for a matrix
pseudo_obs(cbind(rnorm(10), rnorm(10)))
```

`rosenblatt` *(Inverse) Rosenblatt transform*

Description

The Rosenblatt transform takes data generated from a model and turns it into independent uniform variates, The inverse Rosenblatt transform computes conditional quantiles and can be used simulate from a stochastic model, see *Details*.

Usage

```
rosenblatt(x, model, cores = 1, randomize_discrete = TRUE)

inverse_rosenblatt(u, model, cores = 1)
```

Arguments

`x`	matrix of evaluation points; must be in $(0, 1)^d$ for copula models.
`model`	a model object; classes currently supported are `bicop_dist()`, `vinecop_dist()`, and `vine_dist()`.
`cores`	if >1, computation is parallelized over `cores` batches (rows of `u`).
`randomize_discrete`	Whether to randomize the transform for discrete variables; see Details.
`u`	matrix of evaluation points; must be in $(0, 1)^d$.

Details

The Rosenblatt transform (Rosenblatt, 1952) $U = T(V)$ of a random vector $V = (V_1, \ldots, V_d)$ F is defined as

$$U_1 = F(V_1), U_2 = F(V_2|V_1), \ldots, U_d = F(V_d|V_1, \ldots, V_{d-1}),$$

where $F(v_k|v_1, \ldots, v_{k-1})$ is the conditional distribution of V_k given $V_1 \ldots, V_{k-1}, k = 2, \ldots, d$. The vector $U = (U_1, \ldots, U_d)$ then contains independent standard uniform variables. The inverse operation

$$V_1 = F^{-1}(U_1), V_2 = F^{-1}(U_2|U_1), \ldots, V_d = F^{-1}(U_d|U_1, \ldots, U_{d-1}),$$

can be used to simulate from a distribution. For any copula F, if U is a vector of independent random variables, $V = T^{-1}(U)$ has distribution F.

The formulas above assume a vine copula model with order $d, \dots, 1$. More generally, `rosenblatt()` returns the variables

$$U_{M[d+1-j,j]} = F(V_{M[d-j+1,j]} | V_{M[d-j,j]}, \dots, V_{M[1,j]}),$$

where M is the structure matrix. Similarly, `inverse_rosenblatt()` returns

$$V_{M[d+1-j,j]} = F^{-1}(U_{M[d-j+1,j]} | U_{M[d-j,j]}, \dots, U_{M[1,j]}).$$

If some variables have atoms, Brockwell (10.1016/j.spl.2007.02.008) proposed a simple randomization scheme to ensure that output is still independent uniform if the model is correct. The transformation reads

$$U_{M[d-j,j]} = W_{d-j} F(V_{M[d-j,j]} | V_{M[d-j-1,j-1]}, \dots, V_{M[0,0]}) + (1 - W_{d-j}) F^{-}(V_{M[d-j,j]} | V_{M[d-j-1,j-1]}, \dots, V_{M[0,0]}),$$

where F^{-} is the left limit of the conditional cdf and $W_1, \dots, W_d$ are are independent standard uniform random variables. This is used by default. If you are interested in the conditional probabilities

$$F(V_{M[d-j,j]} | V_{M[d-j-1,j-1]}, \dots, V_{M[0,0]}),$$

set `randomize_discrete = FALSE`.

Examples

```
# simulate data with some dependence
x <- replicate(3, rnorm(200))
x[, 2:3] <- x[, 2:3] + x[, 1]
pairs(x)

# estimate a vine distribution model
fit <- vine(x, copula_controls = list(family_set = "par"))

# transform into independent uniforms
u <- rosenblatt(x, fit)
pairs(u)

# inversion
pairs(inverse_rosenblatt(u, fit))

# works similarly for vinecop models
vc <- fit$copula
rosenblatt(pseudo_obs(x), vc)
```

`rvinecopulib` *High Performance Algorithms for Vine Copula Modeling*

Description

Provides an interface to 'vinecopulib', a C++ library for vine copula modeling based on 'Boost' and 'Eigen'. The 'rvinecopulib' package implements the core features of the popular 'VineCopula' package, in particular inference algorithms for both vine copula and bivariate copula models. Advantages over 'VineCopula' are a sleeker and more modern API, improved performances, especially in high dimensions, nonparametric and multi-parameter families. The 'rvinecopulib' package includes 'vinecopulib' as header-only C++ library (currently version 0.6.2). Thus users do not need to install 'vinecopulib' itself in order to use 'rvinecopulib'. Since their initial releases, 'vinecopulib' is licensed under the MIT License, and 'rvinecopulib' is licensed under the GNU GPL version 3.

Author(s)

Thomas Nagler, Thibault Vatter

See Also

Useful links:

- https://vinecopulib.github.io/rvinecopulib/
- Report bugs at https://github.com/vinecopulib/rvinecopulib/issues

Examples

```
## bicop_dist objects
bicop_dist("gaussian", 0, 0.5)
str(bicop_dist("gauss", 0, 0.5))
bicop <- bicop_dist("clayton", 90, 3)

## bicop objects
u <- rbicop(500, "gauss", 0, 0.5)
fit1 <- bicop(u, family = "par")
fit1

## vinecop_dist objects
## specify pair-copulas
bicop <- bicop_dist("bb1", 90, c(3, 2))
pcs <- list(
  list(bicop, bicop), # pair-copulas in first tree
  list(bicop) # pair-copulas in second tree
)
## specify R-vine matrix
mat <- matrix(c(1, 2, 3, 1, 2, 0, 1, 0, 0), 3, 3)
## build the vinecop_dist object
vc <- vinecop_dist(pcs, mat)
summary(vc)

## vinecop objects
u <- sapply(1:3, function(i) runif(50))
vc <- vinecop(u, family = "par")
summary(vc)
```

```
## vine_dist objects
vc <- vine_dist(list(list(distr = "norm")), pcs, mat)
summary(vc)

## vine objects
x <- sapply(1:3, function(i) rnorm(50))
vc <- vine(x, copula_controls = list(family_set = "par"))
summary(vc)
```

`rvine_structure` *R-vine structure*

Description

R-vine structures are compressed representations encoding the tree structure of the vine, i.e. the conditioned/conditioning variables of each edge. The functions `[cvine_structure()]` or `[dvine_structure()]` give a simpler way to construct C-vines (every tree is a star) and D-vines (every tree is a path), respectively (see *Examples*).

Usage

```
rvine_structure(order, struct_array = list(), is_natural_order = FALSE)

cvine_structure(order, trunc_lvl = Inf)

dvine_structure(order, trunc_lvl = Inf)

rvine_matrix(matrix)
```

Arguments

`order`	a vector of positive integers.
`struct_array`	a list of vectors of positive integers. The vectors represent rows of the r-rvine structure and the number of elements have to be compatible with the `order` vector. If empty, the model is 0-truncated.
`is_natural_order`	whether `struct_array` is assumed to be provided in natural order already (a structure is in natural order if the anti- diagonal is 1, .., d from bottom left to top right).
`trunc_lvl`	the truncation level
`matrix`	an R-vine matrix, see *Details*.

Details

The R-vine structure is essentially a lower-triangular matrix/triangular array, with a notation that differs from the one in the VineCopula package. An example array is

```
4 4 4 4
3 3 3
2 2
1
```

which encodes the following pair-copulas:

tree	edge	pair-copulas
0	0	`(1, 4)`
	1	`(2, 4)`
	2	`(3, 4)`
1	0	`(1, 3; 4)`
	1	`(2, 3; 4)`
2	0	`(1, 2; 3, 4)`

An R-vine structure can be converted to an R-vine matrix using `as_rvine_matrix()`, which encodes the same model with a square matrix filled with zeros. For instance, the matrix corresponding to the structure above is:

```
4 4 4 4
3 3 3 0
2 2 0 0
1 0 0 0
```

Similarly, an R-vine matrix can be converted to an R-vine structure using `as_rvine_structure()`.

Denoting by `M[i, j]` the array entry in row `i` and column `j` (the pair-copula index for edge `e` in tree `t` of a `d` dimensional vine is `(M[d + 1 - e, e], M[t, e]; M[t - 1, e], ..., M[1, e])`. Less formally,

1. Start with the counter-diagonal element of column `e` (first conditioned variable).
2. Jump up to the element in row `t` (second conditioned variable).
3. Gather all entries further up in column `e` (conditioning set).

Internally, the diagonal is stored separately from the off-diagonal elements, which are stored as a triangular array. For instance, the off-diagonal elements off the structure above are stored as

```
4 4 4
3 3
2
```

for the structure above. The reason is that it allows for parsimonious representations of truncated models. For instance, the 2-truncated model is represented by the same diagonal and the following truncated triangular array:

```
4 4 4
3 3
```

A valid R-vine structure or matrix must satisfy several conditions which are checked when `rvine_structure()`, `rvine_matrix()`, or some coercion methods (see `as_rvine_structure()` and `as_rvine_matrix()` are called:

1. It can only contain numbers between 1 and d (and additionally zeros for R-vine matrices).
2. The anti-diagonal must contain the numbers 1, ..., d.
3. The anti-diagonal entry of a column must not be contained in any column further to the right.
4. The entries of a column must be contained in all columns to the left.
5. The proximity condition must hold: For all t = 1, ..., d - 2 and e = 1, ..., d - t there must exist an index j > d, such that `(M[t, e], {M[1, e], ..., M[t - 1, e]})` equals either `(M[d + 1 - j, j], {M[1, j], ..., M[t - 1, j]})` or `(M[t - 1, j], {M[d + 1 - j, j], M[1, j], ..., M[t - 2, j`

Condition 5 already implies conditions 2-4, but is more difficult to check by hand.

Value

Either an `rvine_structure` or an `rvine_matrix`.

See Also

`as_rvine_structure()`, `as_rvine_matrix()`, `plot.rvine_structure()`, `plot.rvine_matrix()`, `rvine_structure_sim()`, `rvine_matrix_sim()`

Examples

```
# R-vine structures can be constructed from the order vector and struct_array
rvine_structure(order = 1:4, struct_array = list(
  c(4, 4, 4),
  c(3, 3),
  2
))

# R-vine matrices can be constructed from standard matrices
mat <- matrix(c(4, 3, 2, 1, 4, 3, 2, 0, 4, 3, 0, 0, 4, 0, 0, 0), 4, 4)
rvine_matrix(mat)

# coerce to R-vine structure
str(as_rvine_structure(mat))

# truncate and construct the R-vine matrix
mat[3, 1] <- 0
rvine_matrix(mat)

# or use directly the R-vine structure constructor
rvine_structure(order = 1:4, struct_array = list(
  c(4, 4, 4),
  c(3, 3)
))

# throws an error
mat[3, 1] <- 5
```

```
try(rvine_matrix(mat))

# C-vine structure
cvine <- cvine_structure(1:5)
cvine
plot(cvine)

# D-vine structure
dvine <- dvine_structure(c(1, 4, 2, 3, 5))
dvine
plot(dvine)
```

`rvine_structure_sim` *Simulate R-vine structures*

Description

Simulates from a uniform distribution over all R-vine structures on d variables. `rvine_structure_sim()` returns an `rvine_structure()` object, `rvine_matrix_sim()` an `rvine_matrix()`.

Usage

```
rvine_structure_sim(d, natural_order = FALSE)

rvine_matrix_sim(d, natural_order = FALSE)
```

Arguments

`d`	the number of variables
`natural_order`	boolean; whether the structures should be in natural order (counter-diagonal is `1:d`).

See Also

`rvine_structure()`, `rvine_matrix()`, `plot.rvine_structure()`, `plot.rvine_matrix()`

Examples

```
rvine_structure_sim(10)

rvine_structure_sim(10, natural_order = TRUE)  # counter-diagonal is 1:d

rvine_matrix_sim(10)
```

truncate_model *Truncate a vine copula model*

Description

Extracts a truncated sub-vine based on a truncation level supplied by user.

Usage

```
truncate_model(object, ...)

## S3 method for class 'rvine_structure'
truncate_model(object, trunc_lvl, ...)

## S3 method for class 'rvine_matrix'
truncate_model(object, trunc_lvl, ...)

## S3 method for class 'vinecop_dist'
truncate_model(object, trunc_lvl, ...)

## S3 method for class 'vine_dist'
truncate_model(object, trunc_lvl, ...)
```

Arguments

`object`	a model object.
`...`	further arguments passed to specific methods.
`trunc_lvl`	tree level after which the vine copula should be truncated.

Details

While a vine model for a `d` dimensional random vector contains at most `d-1` nested trees, this function extracts a sub-model based on a given truncation level.

For instance, `truncate_model(object, 1)` results in a 1-truncated vine (i.e., a vine with a single tree). Similarly `truncate_model(object, 2)` results in a 2-truncated vine (i.e., a vine with two trees). Note that `truncate_model(truncate_model(object, 1), 2)` returns a 1-truncated vine.

Examples

```
# specify pair-copulas
bicop <- bicop_dist("bb1", 90, c(3, 2))
pcs <- list(
  list(bicop, bicop), # pair-copulas in first tree
  list(bicop) # pair-copulas in second tree
)

# specify R-vine matrix
```

```
mat <- matrix(c(1, 2, 3, 1, 2, 0, 1, 0, 0), 3, 3)

# set up vine structure
structure <- as_rvine_structure(mat)

# truncate the model
truncate_model(structure, 1)

# set up vine copula model
vc <- vinecop_dist(pcs, mat)

# truncate the model
truncate_model(vc, 1)
```

---

vine *Vine copula models*

Description

Automated fitting or creation of custom vine copula models

Usage

```
vine(
  data,
  margins_controls = list(mult = NULL, xmin = NaN, xmax = NaN, bw = NA, deg = 2),
  copula_controls = list(family_set = "all", structure = NA, par_method = "mle",
   nonpar_method = "constant", mult = 1, selcrit = "aic", psi0 = 0.9, presel = TRUE,
   allow_rotations = TRUE, trunc_lvl = Inf, tree_crit = "tau", threshold = 0, keep_data
    = FALSE, show_trace = FALSE, cores = 1, tree_algorithm = "mst_prim"),
  weights = numeric(),
  keep_data = FALSE,
  cores = 1
)

vine_dist(margins, pair_copulas, structure)
```

Arguments

`data`	a matrix or data.frame. Discrete variables have to be declared as `ordered()`.
`margins_controls`	a list with arguments to be passed to `kde1d::kde1d()`. Currently, there can be • `mult` numeric vector of length one or d; all bandwidths for marginal kernel density estimation are multiplied with `mult`. Defaults to `log(1 + d)` where d is the number of variables after applying `rvinecopulib:::expand_factors()`. • `xmin` numeric vector of length d; see `kde1d::kde1d()`. • `xmax` numeric vector of length d; see `kde1d::kde1d()`.

- type numeric vector of length d; see kde1d::kde1d().
- bw numeric vector of length d; see kde1d::kde1d().
- deg numeric vector of length one or d; kde1d::kde1d().

copula_controls	a list with arguments to be passed to vinecop().
weights	optional vector of weights for each observation.
keep_data	whether the original data should be stored; if you want to store the pseudo-observations used for fitting the copula, use the copula_controls argument.
cores	the number of cores to use for parallel computations.
margins	A list with with each element containing the specification of a marginal stats::Distributions. Each marginal specification should be a list with containing at least the distribution family ("distr") and optionally the parameters, e.g. list(list(distr = "norm"), list(distr = "norm", mu = 1), list(distr = "beta", shape1 = 1, shape2 = 1)). Note that parameters that have no default values have to be provided. Furthermore, if margins has length one, it will be recycled for every component.
pair_copulas	A nested list of 'bicop_dist' objects, where pair_copulas[[t]][[e]] corresponds to the pair-copula at edge e in tree t.
structure	an rvine_structure object, namely a compressed representation of the vine structure, or an object that can be coerced into one (see rvine_structure() and as_rvine_structure()). The dimension must be length(pair_copulas[[1]]) + 1.

Details

vine_dist() creates a vine copula by specifying the margins, a nested list of bicop_dist objects and a quadratic structure matrix.

vine() provides automated fitting for vine copula models. margins_controls is a list with the same parameters as kde1d::kde1d() (except for x). copula_controls is a list with the same parameters as vinecop() (except for data).

Value

Objects inheriting from vine_dist for vine_dist(), and vine and vine_dist for vine().

Objects from the vine_dist class are lists containing:

- margins, a list of marginals (see below).
- copula, an object of the class vinecop_dist, see vinecop_dist().

For objects from the vine class, copula is also an object of the class vine, see vinecop(). Additionally, objects from the vine class contain:

- margins_controls, a list with the set of fit controls that was passed to kde1d::kde1d() when estimating the margins.
- copula_controls, a list with the set of fit controls that was passed to vinecop() when estimating the copula.

- data (optionally, if keep_data = TRUE was used), the dataset that was passed to vine().
- nobs, an integer containing the number of observations that was used to fit the model.

Concerning margins:

- For objects created with vine_dist(), it simply corresponds to the margins argument.
- For objects created with vine(), it is a list of objects of class kde1d, see kde1d::kde1d().

Examples

```
# specify pair-copulas
bicop <- bicop_dist("bb1", 90, c(3, 2))
pcs <- list(
  list(bicop, bicop), # pair-copulas in first tree
  list(bicop) # pair-copulas in second tree
)

# specify R-vine matrix
mat <- matrix(c(1, 2, 3, 1, 2, 0, 1, 0, 0), 3, 3)

# set up vine copula model with Gaussian margins
vc <- vine_dist(list(list(distr = "norm")), pcs, mat)

# show model
summary(vc)

# simulate some data
x <- rvine(50, vc)

# estimate a vine copula model
fit <- vine(x, copula_controls = list(family_set = "par"))
summary(fit)

## model for discrete data
x <- as.data.frame(x)
x[, 1] <- ordered(round(x[, 1]), levels = seq.int(-5, 5))
fit_disc <- vine(x, copula_controls = list(family_set = "par"))
summary(fit_disc)
```

vinecop *Fitting vine copula models*

Description

Automated fitting and model selection for vine copula models with continuous or discrete data. Selection of the structure is performed using the algorithm of Dissmann et al. (2013).

Usage

```
vinecop(
  data,
  var_types = rep("c", NCOL(data)),
  family_set = "all",
  structure = NA,
  par_method = "mle",
  nonpar_method = "constant",
  mult = 1,
  selcrit = "aic",
  weights = numeric(),
  psi0 = 0.9,
  presel = TRUE,
  allow_rotations = TRUE,
  trunc_lvl = Inf,
  tree_crit = "tau",
  threshold = 0,
  keep_data = FALSE,
  vinecop_object = NULL,
  show_trace = FALSE,
  cores = 1,
  tree_algorithm = "mst_prim"
)
```

Arguments

`data`	a matrix or data.frame with at least two columns, containing the (pseudo-)observations for the two variables (copula data should have approximately uniform margins). More columns are required for discrete models, see *Details*.
`var_types`	variable types, a length d vector; e.g., `c("c", "c")` for two continuous variables, or `c("c", "d")` for first variable continuous and second discrete.
`family_set`	a character vector of families; see `bicop()` for additional options.
`structure`	an `rvine_structure` object, namely a compressed representation of the vine structure, or an object that can be coerced into one (see `rvine_structure()` and `as_rvine_structure()`). The dimension must be `length(pair_copulas[[1]]) + 1`; `structure = NA` performs automatic selection based on Dissman's algorithm. See *Details* for partial selection of the structure.
`par_method`	the estimation method for parametric models, either `"mle"` for maximum likelihood or `"itau"` for inversion of Kendall's tau (only available for one-parameter families and `"t"`.
`nonpar_method`	the estimation method for nonparametric models, either `"constant"` for the standard transformation estimator, or `"linear"`/`"quadratic"` for the local-likelihood approximations of order one/two.
`mult`	multiplier for the smoothing parameters of nonparametric families. Values larger than 1 make the estimate more smooth, values less than 1 less smooth.
`selcrit`	criterion for family selection, either `"loglik"`, `"aic"`, `"bic"`, `"mbic"`. For `vinecop()` there is the additional option `"mbicv"`.

`weights`	optional vector of weights for each observation.
`psi0`	prior probability of a non-independence copula (only used for `selcrit = "mbic"` and `selcrit = "mbicv"`).
`presel`	whether the family set should be thinned out according to symmetry characteristics of the data.
`allow_rotations`	whether to allow rotations of the copula.
`trunc_lvl`	the truncation level of the vine copula; `Inf` means no truncation, `NA` indicates that the truncation level should be selected automatically by `mBICV()`.
`tree_crit`	the criterion for tree selection, one of `"tau"`, `"rho"`, `"hoeffd"`, `"mcor"`, or `"joe"` for Kendall's τ, Spearman's ρ, Hoeffding's D, maximum correlation, or logarithm of the partial correlation, respectively.
`threshold`	for thresholded vine copulas; `NA` indicates that the threshold should be selected automatically by `mBICV()`.
`keep_data`	whether the data should be stored (necessary for using `fitted()`).
`vinecop_object`	a `vinecop` object to be updated; if provided, only the parameters are fit; structure and families are kept the same.
`show_trace`	logical; whether a trace of the fitting progress should be printed.
`cores`	number of cores to use; if more than 1, estimation of pair copulas within a tree is done in parallel.
`tree_algorithm`	The algorithm for building the spanning tree (`"mst_prim"`, `"mst_kruskal"`, `"random_weighted"`, or `"random_unweighted"`) during the tree-wise structure selection. `"mst_prim"` and `"mst_kruskal"` use Prim's and Kruskal's algorithms respectively to select the maximum spanning tree, maximizing the sum of the edge weights (i.e., `tree_criterion`). `"random_weighted"` and `"random_unweighted"` use Wilson's algorithm to generate a random spanning tree, either with probability proportional to the product of the edge weights (weighted) or uniformly (unweighted).

Details

Missing data:

If there are missing data (i.e., `NA` entries), incomplete observations are discarded before fitting a pair-copula. This is done on a pair-by-pair basis so that the maximal available information is used.

Discrete variables:

The dependence measures used to select trees (default: Kendall's tau) are corrected for ties (see wdm::wdm).

Let `n` be the number of observations and `d` the number of variables. When at least one variable is discrete, two types of "observations" are required in `data`: the first `n x d` block contains realizations of $F_{X_j}(X_j)$. The second `n x d` block contains realizations of $F_{X_j}(X_j^-)$. The minus indicates a left-sided limit of the cdf. For, e.g., an integer-valued variable, it holds $F_{X_j}(X_j^-) = F_{X_j}(X_j - 1)$. For continuous variables the left limit and the cdf itself coincide. Respective columns can be omitted in the second block.

Structure selection:

Selection of the structure is performed using the algorithm of Dissmann, J. F., E. C. Brechmann, C. Czado, and D. Kurowicka (2013). *Selecting and estimating regular vine copulae and application to financial returns.* Computational Statistics & Data Analysis, 59 (1), 52-69. The dependence measure used to select trees (default: Kendall's tau) is corrected for ties and can be changed using the `tree_criterion` argument, which can be set to `"tau"`, `"rho"` or `"hoeffd"`. Both Prim's (default: `"mst_prim"`) and Kruskal's ()`"mst_kruskal"`) algorithms are available through `tree_algorithm` to set the maximum spanning tree selection algorithm. An alternative to the maximum spanning tree selection is to use random spanning trees, which can be selected using `controls.tree_algorithm` and come in two flavors, both using Wilson's algorithm loop erased random walks:

- "random_weighted"‘ generates a random spanning tree with probability proportional to the product of the weights (i.e., the dependence) of the edges in the tree.
- "random_unweighted"‘ generates a random spanning tree uniformly over all spanning trees satisfying the proximity condition.

Partial structure selection:

It is possible to fix the vine structure only in the first trees and select the remaining ones automatically. To specify only the first `k` trees, supply a `k`-truncated `rvine_structure()` or `rvine_matrix()`. All trees up to `trunc_lvl` will then be selected automatically.

Value

Objects inheriting from `vinecop` and `vinecop_dist` for `vinecop()`. In addition to the entries provided by `vinecop_dist()`, there are:

- `threshold`, the (set or estimated) threshold used for thresholding the vine.
- `data` (optionally, if `keep_data = TRUE` was used), the dataset that was passed to `vinecop()`.
- `controls`, a `list` with fit controls that was passed to `vinecop()`.
- `nobs`, the number of observations that were used to fit the model.

References

Dissmann, J. F., E. C. Brechmann, C. Czado, and D. Kurowicka (2013). *Selecting and estimating regular vine copulae and application to financial returns.* Computational Statistics & Data Analysis, 59 (1), 52-69.

See Also

`vinecop()`, `dvinecop()`, `pvinecop()`, `rvinecop()`, `plot.vinecop()`, `contour.vinecop()`

Examples

```
## simulate dummy data
x <- rnorm(30) * matrix(1, 30, 5) + 0.5 * matrix(rnorm(30 * 5), 30, 5)
u <- pseudo_obs(x)

## fit and select the model structure, family and parameters
fit <- vinecop(u)
```

```
summary(fit)
plot(fit)
contour(fit)

## select by log-likelihood criterion from one-paramter families
fit <- vinecop(u, family_set = "onepar", selcrit = "bic")
summary(fit)

## 1-truncated, Gaussian D-vine
fit <- vinecop(u, structure = dvine_structure(1:5), family = "gauss", trunc_lvl = 1)
plot(fit)
contour(fit)

## Partial structure selection with only first tree specified
structure <- rvine_structure(order = 1:5, list(rep(5, 4)))
structure
fit <- vinecop(u, structure = structure, family = "gauss")
plot(fit)

## Model for discrete data
x <- qpois(u, 1)  # transform to Poisson margins
# we require two types of observations (see Details)
u_disc <- cbind(ppois(x, 1), ppois(x - 1, 1))
fit <- vinecop(u_disc, var_types = rep("d", 5))

## Model for mixed data
x <- qpois(u[, 1], 1)  # transform first variable to Poisson margin
# we require two types of observations (see Details)
u_disc <- cbind(ppois(x, 1), u[, 2:5], ppois(x - 1, 1))
fit <- vinecop(u_disc, var_types = c("d", rep("c", 4)))
```

---

vinecop_dist *Vine copula models*

Description

Create custom vine copula models by specifying the pair-copulas, structure, and variable types.

Usage

```
vinecop_dist(pair_copulas, structure, var_types = rep("c", dim(structure)[1]))
```

Arguments

`pair_copulas`	A nested list of '`bicop_dist()`' objects, where `pair_copulas[[t]][[e]]` corresponds to the pair-copula at edge `e` in tree `t`.
`structure`	an `rvine_structure` object, namely a compressed representation of the vine structure, or an object that can be coerced into one (see `rvine_structure()` and `as_rvine_structure()`). The dimension must be `length(pair_copulas[[1]])`

+ 1; structure = NA performs automatic selection based on Dissman's algorithm. See *Details* for partial selection of the structure.

var_types: variable types, a length d vector; e.g., c("c", "c") for two continuous variables, or c("c", "d") for first variable continuous and second discrete.

Value

Object of class vinecop_dist, i.e., a list containing:

- pair_copulas, a list of lists. Each element of pair_copulas corresponds to a tree, which is itself a list of bicop_dist() objects.
- structure, a compressed representation of the vine structure, or an object that can be coerced into one (see rvine_structure() and as_rvine_structure()).
- npars, a numeric with the number of (effective) parameters.
- var_types the variable types.

See Also

rvine_structure(), rvine_matrix(), vinecop(), plot.vinecop_dist(), contour.vinecop_dist(), dvinecop(), pvinecop(), rvinecop()

Examples

```
# specify pair-copulas
bicop <- bicop_dist("bb1", 90, c(3, 2))
pcs <- list(
  list(bicop, bicop), # pair-copulas in first tree
  list(bicop) # pair-copulas in second tree
)

# specify R-vine matrix
mat <- matrix(c(1, 2, 3, 1, 2, 0, 1, 0, 0), 3, 3)

# set up vine copula model
vc <- vinecop_dist(pcs, mat)

# visualization
plot(vc)
contour(vc)

# simulate from the model
pairs(rvinecop(200, vc))
```

`vinecop_distributions` *Vine copula distributions*

Description

Density, distribution function and random generation for the vine copula distribution.

Usage

```
dvinecop(u, vinecop, cores = 1)

pvinecop(u, vinecop, n_mc = 10^4, cores = 1)

rvinecop(n, vinecop, qrng = FALSE, cores = 1)
```

Arguments

`u`	matrix of evaluation points; must contain at least d columns, where d is the number of variables in the vine. More columns are required for discrete models, see *Details*.
`vinecop`	an object of class `"vinecop_dist"`.
`cores`	number of cores to use; if larger than one, computations are done in parallel on `cores` batches .
`n_mc`	number of samples used for quasi Monte Carlo integration.
`n`	number of observations.
`qrng`	if `TRUE`, generates quasi-random numbers using the multivariate Generalized Halton sequence up to dimension 300 and the Generalized Sobol sequence in higher dimensions (default `qrng` = `FALSE`).

Details

See `vinecop()` for the estimation and construction of vine copula models.

The copula density is defined as joint density divided by marginal densities, irrespective of variable types.

Discrete variables:

When at least one variable is discrete, two types of "observations" are required in `u`: the first $n\ x\ d$ block contains realizations of $F_{X_j}(X_j)$. The second $n\ x\ d$ block contains realizations of $F_{X_j}(X_j^-)$. The minus indicates a left-sided limit of the cdf. For, e.g., an integer-valued variable, it holds $F_{X_j}(X_j^-) = F_{X_j}(X_j - 1)$. For continuous variables the left limit and the cdf itself coincide. Respective columns can be omitted in the second block.

Value

`dvinecop()` gives the density, `pvinecop()` gives the distribution function, and `rvinecop()` generates random deviates.

The length of the result is determined by `n` for `rvinecop()`, and the number of rows in `u` for the other functions.

The `vinecop` object is recycled to the length of the result.

See Also

`vinecop_dist()`, `vinecop()`, `plot.vinecop()`, `contour.vinecop()`

Examples

```
## simulate dummy data
x <- rnorm(30) * matrix(1, 30, 5) + 0.5 * matrix(rnorm(30 * 5), 30, 5)
u <- pseudo_obs(x)

## fit a model
vc <- vinecop(u, family = "clayton")

# simulate from the model
u <- rvinecop(100, vc)
pairs(u)

# evaluate the density and cdf
dvinecop(u[1, ], vc)
pvinecop(u[1, ], vc)

## Discrete models
vc$var_types <- rep("d", 5)  # convert model to discrete

# with discrete data we need two types of observations (see Details)
x <- qpois(u, 1)  # transform to Poisson margins
u_disc <- cbind(ppois(x, 1), ppois(x - 1, 1))

dvinecop(u_disc[1:5, ], vc)
pvinecop(u_disc[1:5, ], vc)

# simulated data always has uniform margins
pairs(rvinecop(200, vc))
```

---

vinecop_predict_and_fitted

Predictions and fitted values for a vine copula model

Description

Predictions of the density and distribution function for a vine copula model.

Usage

```
## S3 method for class 'vinecop'
predict(object, newdata, what = "pdf", n_mc = 10^4, cores = 1, ...)

## S3 method for class 'vinecop'
fitted(object, what = "pdf", n_mc = 10^4, cores = 1, ...)
```

Arguments

`object`	a `vinecop` object.
`newdata`	points where the fit shall be evaluated.
`what`	what to predict, either `"pdf"` or `"cdf"`.
`n_mc`	number of samples used for quasi Monte Carlo integration when `what = "cdf"`.
`cores`	number of cores to use; if larger than one, computations are done in parallel on `cores` batches.
`...`	unused.

Details

`fitted()` can only be called if the model was fit with the `keep_data = TRUE` option.

Discrete variables:

When at least one variable is discrete, two types of "observations" are required in `newdata`: the first $n\ x\ d$ block contains realizations of $F_{X_j}(X_j)$. The second $n\ x\ d$ block contains realizations of $F_{X_j}(X_j^-)$. The minus indicates a left-sided limit of the cdf. For, e.g., an integer-valued variable, it holds $F_{X_j}(X_j^-) = F_{X_j}(X_j - 1)$. For continuous variables the left limit and the cdf itself coincide. Respective columns can be omitted in the second block.

Value

`fitted()` and `predict()` have return values similar to `dvinecop()` and `pvinecop()`.

Examples

```
u <- sapply(1:5, function(i) runif(50))
fit <- vinecop(u, family = "par", keep_data = TRUE)
all.equal(predict(fit, u), fitted(fit), check.environment = FALSE)
```

vine_distributions *Vine based distributions*

Description

Density, distribution function and random generation for the vine based distribution.

Usage

```
dvine(x, vine, cores = 1)

pvine(x, vine, n_mc = 10^4, cores = 1)

rvine(n, vine, qrng = FALSE, cores = 1)
```

Arguments

x	evaluation points, either a length d vector or a d-column matrix, where d is the number of variables in the vine.
vine	an object of class "vine_dist".
cores	number of cores to use; if larger than one, computations are done in parallel on cores batches .
n_mc	number of samples used for quasi Monte Carlo integration.
n	number of observations.
qrng	if TRUE, generates quasi-random numbers using the multivariate Generalized Halton sequence up to dimension 300 and the Generalized Sobol sequence in higher dimensions (default qrng = FALSE).

Details

See vine for the estimation and construction of vine models. Here, the density, distribution function and random generation for the vine distributions are standard.

The functions are based on `dvinecop()`, `pvinecop()` and `rvinecop()` for vinecop objects, and either `kde1d::dkde1d()`, `kde1d::pkde1d()` and `kde1d::qkde1d()` for estimated vines (i.e., output of `vine()`), or the standard *d/p/q-xxx* from stats::Distributions for custom vines (i.e., output of `vine_dist()`).

Value

`dvine()` gives the density, `pvine()` gives the distribution function, and `rvine()` generates random deviates.

The length of the result is determined by n for `rvine()`, and the number of rows in u for the other functions.

The `vine` object is recycled to the length of the result.

Examples

```
# specify pair-copulas
bicop <- bicop_dist("bb1", 90, c(3, 2))
pcs <- list(
  list(bicop, bicop), # pair-copulas in first tree
  list(bicop) # pair-copulas in second tree
)

# set up vine copula model
mat <- rvine_matrix_sim(3)
vc <- vine_dist(list(list(distr = "norm")), pcs, mat)

# simulate from the model
x <- rvine(200, vc)
pairs(x)

# evaluate the density and cdf
dvine(x[1, ], vc)
pvine(x[1, ], vc)
```

`vine_predict_and_fitted`

Predictions and fitted values for a vine copula model

Description

Predictions of the density and distribution function for a vine copula model.

Usage

```
## S3 method for class 'vine'
predict(object, newdata, what = "pdf", n_mc = 10^4, cores = 1, ...)

## S3 method for class 'vine'
fitted(object, what = "pdf", n_mc = 10^4, cores = 1, ...)
```

Arguments

`object`	a `vine` object.
`newdata`	points where the fit shall be evaluated.
`what`	what to predict, either `"pdf"` or `"cdf"`.
`n_mc`	number of samples used for quasi Monte Carlo integration when `what = "cdf"`.
`cores`	number of cores to use; if larger than one, computations are done in parallel on `cores` batches .
`...`	unused.

Value

fitted() and predict() have return values similar to dvine() and pvine().

Examples

```
x <- sapply(1:5, function(i) rnorm(50))
fit <- vine(x, copula_controls = list(family_set = "par"), keep_data = TRUE)
all.equal(predict(fit, x), fitted(fit), check.environment = FALSE)
```

Index

∗ **package**
  rvinecopulib, 23
∗ **plot**
  plot.bicop_dist, 17
  plot.vinecop_dist, 19

as.bicop, 2
as_rvine_matrix *(*as_rvine_structure*)*, 3
as_rvine_matrix(), *26*, *27*
as_rvine_structure, 3
as_rvine_structure(), *26*, *27*, *31*, *33*, *36*, *37*

bicop, 5, *10*, *14*
bicop(), *7*, *11*, *33*
bicop_dist, 8, *11*, *17*
bicop_dist(), *7*, *9*, *11*, *17*, *18*, *36*, *37*
bicop_distributions, 9
bicop_predict_and_fitted, 11

contour.bicop, *20*
contour.bicop *(*plot.bicop_dist*)*, 17
contour.bicop(), *7*, *9*
contour.bicop_dist *(*plot.bicop_dist*)*, 17
contour.vinecop *(*plot.vinecop_dist*)*, 19
contour.vinecop(), *35*, *39*
contour.vinecop_dist *(*plot.vinecop_dist*)*, 19
contour.vinecop_dist(), *37*
cvine_structure *(*rvine_structure*)*, 25

dbicop *(*bicop_distributions*)*, 9
dbicop(), *7*, *9*, *12*
dbicop_dist *(*bicop_distributions*)*, 9
dvine *(*vine_distributions*)*, 41
dvine(), *43*
dvine_dist *(*vine_distributions*)*, 41
dvine_structure *(*rvine_structure*)*, 25
dvinecop *(*vinecop_distributions*)*, 38
dvinecop(), *35*, *37*, *40*, *41*
dvinecop_dist *(*vinecop_distributions*)*, 38

emp_cdf, 12

fitted(), *6*, *34*
fitted.bicop *(*bicop_predict_and_fitted*)*, 11
fitted.vine *(*vine_predict_and_fitted*)*, 42
fitted.vinecop *(*vinecop_predict_and_fitted*)*, 39

get_all_families *(*getters*)*, 13
get_all_ktaus *(*getters*)*, 13
get_all_pair_copulas *(*getters*)*, 13
get_all_parameters *(*getters*)*, 13
get_family *(*getters*)*, 13
get_family(), *14*
get_ktau *(*getters*)*, 13
get_ktau(), *14*
get_matrix, *14*
get_matrix *(*getters*)*, 13
get_pair_copula *(*getters*)*, 13
get_pair_copula(), *14*
get_parameters *(*getters*)*, 13
get_parameters(), *14*
get_structure, *14*
get_structure *(*getters*)*, 13
getters, 13
graphics::contour(), *18*

hbicop *(*bicop_distributions*)*, 9
hbicop(), *7*, *9*, *12*
hbicop_dist *(*bicop_distributions*)*, 9

inverse_rosenblatt *(*rosenblatt*)*, 22
is.rvine_matrix *(*rvine_structure*)*, 25
is.rvine_structure *(*rvine_structure*)*, 25

kde1d::dkde1d(), *41*
kde1d::kde1d(), *30–32*
kde1d::pkde1d(), *41*

kde1d::qkde1d(), *41*
ktau_to_par *(*par_to_ktau*)*, 17

lattice::wireframe(), *18*

mBICV, 15
mBICV(), *6*, *34*

pairs_copula_data, 16
par_to_ktau, 17
pbicop *(*bicop_distributions*)*, 9
pbicop(), *7*, *9*, *12*
pbicop_dist *(*bicop_distributions*)*, 9
plot.bicop, *19*, *20*
plot.bicop *(*plot.bicop_dist*)*, 17
plot.bicop(), *7*, *9*
plot.bicop_dist, 17
plot.rvine_matrix
  *(*plot.rvine_structure*)*, 18
plot.rvine_matrix(), *27*, *28*
plot.rvine_structure, 18
plot.rvine_structure(), *27*, *28*
plot.vinecop *(*plot.vinecop_dist*)*, 19
plot.vinecop(), *35*, *39*
plot.vinecop_dist, 19
plot.vinecop_dist(), *37*
predict.bicop
  *(*bicop_predict_and_fitted*)*, 11
predict.bicop_dist
  *(*bicop_predict_and_fitted*)*, 11
predict.vine *(*vine_predict_and_fitted*)*,
  42
predict.vinecop
  *(*vinecop_predict_and_fitted*)*,
  39
pseudo_obs, 21
pvine *(*vine_distributions*)*, 41
pvine(), *43*
pvine_dist *(*vine_distributions*)*, 41
pvinecop *(*vinecop_distributions*)*, 38
pvinecop(), *35*, *37*, *40*, *41*
pvinecop_dist *(*vinecop_distributions*)*,
  38

rank(), *21*
rbicop *(*bicop_distributions*)*, 9
rbicop(), *7*, *9*
rbicop_dist *(*bicop_distributions*)*, 9
rosenblatt, 22
rvine *(*vine_distributions*)*, 41
rvine_dist *(*vine_distributions*)*, 41
rvine_matrix *(*rvine_structure*)*, 25
rvine_matrix(), *3*, *4*, *27*, *28*, *37*
rvine_matrix_sim *(*rvine_structure_sim*)*,
  28
rvine_matrix_sim(), *27*
rvine_structure, *14*, 25
rvine_structure(), *3*, *4*, *27*, *28*, *31*, *33*, *36*,
  37
rvine_structure_sim, 28
rvine_structure_sim(), *27*
rvinecop *(*vinecop_distributions*)*, 38
rvinecop(), *35*, *37*, *41*
rvinecop_dist *(*vinecop_distributions*)*,
  38
rvinecopulib, 23
rvinecopulib-package *(*rvinecopulib*)*, 23

stats::Distributions, *31*, *41*

truncate_model, 29

vine, 30, *41*
vine(), *31*, *32*, *41*
vine_dist *(*vine*)*, 30
vine_dist(), *31*, *32*, *41*
vine_distributions, 41
vine_predict_and_fitted, 42
vinecop, 32, *41*
vinecop(), *31*, *35*, *37–39*
vinecop_dist, *20*, 36
vinecop_dist(), *31*, *35*, *39*
vinecop_distributions, 38
vinecop_predict_and_fitted, 39

wdm::wdm, *34*