

Package ‘rvinecopulib’

September 20, 2026

Type Package

Title High Performance Algorithms for Vine Copula Modeling

Version 1.0.0.1.0

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 1.0.0). 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

SystemRequirements C++17

Depends R (>= 4.3.0)

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

Suggests igraph, ggplot2, ggraph, knitr, rmarkdown, testthat,
univariateML (>= 1.5.0)

VignetteBuilder knitr

LinkingTo BH (>= 1.75.0-0), Rcpp, RcppEigen, RcppThread (>= 2.1.2),
wdm (>= 0.3.0)

BugReports <https://github.com/vinecopulib/rvinecopulib/issues>

URL <https://vinecopulib.github.io/rvinecopulib/>

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

Author Thomas Nagler [aut, cre],
 Thibault Vatter [aut]
Maintainer Thomas Nagler <info@vinecopulib.org>
Repository CRAN
Date/Publication 2026-09-20 09:00:03 UTC

Contents

as.bicop	3
as_margin	3
as_rvine_structure	4
bicop	6
bicop_dependence	8
bicop_dist	9
bicop_distributions	11
bicop_predict_and_fitted	14
emp_cdf	15
getters	16
kde1d_family	17
margin_dist	18
margin_family	19
margin_family_protocol	20
margin_protocol	21
mBICV	22
pairs_copula_data	23
parameter_uncertainty	23
par_to_ktau	26
plot.bicop_dist	26
plot.rvine_structure	27
plot.vinecop_dist	28
pseudo_obs	30
rosenblatt	31
rvinecopulib	33
rvine_structure	34
rvine_structure_sim	37
stats_margin	38
truncate_model	39
univariateML_family	40
vine	40
vinecop	44
vinecop_dist	48
vinecop_distributions	49
vinecop_predict_and_fitted	52
vine_distributions	54
vine_predict_and_fitted	55
zero_inflated	56

as.bicop	<i>Convert list to bicop object</i>
----------	-------------------------------------

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_margin	<i>Normalize an object to the fitted-margin protocol</i>
-----------	--

Description

as_margin() validates fitted margins and adapts the legacy list(distr = ...) representation through [stats_margin\(\)](#).

Usage

```
as_margin(margin)
```

Arguments

margin	a fitted margin or legacy stats distribution list.
--------	--

Value

A validated fitted margin.

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 <i>Details</i>).
...	Other arguments passed on to individual methods.
validate	When ‘TRUE’, verifies that the input is a valid rvine-structure (see <i>Details</i>). 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", "d"),
  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 <i>Details</i> .
-------------	---

var_types	variable types, a length 2 vector; e.g., <code>c("c", "c")</code> for both continuous (default), or <code>c("c", "d")</code> for first variable continuous and second discrete.
family_set	a character vector of families; see <i>Details</i> for additional options.
par_method	the estimation method for parametric models, either <code>"mle"</code> for maximum likelihood or <code>"itau"</code> for inversion of Kendall's tau (only available for one-parameter families and <code>"t"</code>).
nonpar_method	the estimation method for nonparametric models, either <code>"constant"</code> for the standard transformation estimator, or <code>"linear"/"quadratic"</code> 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 <code>"loglik"</code> , <code>"aic"</code> , <code>"bic"</code> , <code>"mbic"</code> . For <code>vinecop()</code> there is the additional option <code>"mbicv"</code> .
weights	optional vector of weights for each observation.
psi0	see <code>mBICV()</code> .
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 <code>fitted()</code>).
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 `"t11"`),
- `"nonparametric"` contains the nonparametric families (`"indep"` and `"t11"`)
- `"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 parametric 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_dependence

Dependence measures of a bivariate copula

Description

Computes the four corner tail-dependence coefficients or Blomqvist's beta for a bivariate copula distribution.

Usage

```
tail_dep(object)

blomqvist_beta(object)
```

Arguments

object a [bicop_dist](#) or fitted [bicop](#) object.

Value

tail_dep() returns a 2 by 2 matrix whose rows refer to the lower and upper tail of the first variable and whose columns refer to the lower and upper tail of the second variable. blomqvist_beta() returns a numeric scalar.

Examples

```
cop <- bicop_dist("clayton", 0, 2)
tail_dep(cop)
blomqvist_beta(cop)
```

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 <i>Details</i> 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"),
  log = FALSE,
  deriv = NULL,
  cores = 1
)

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

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

## S3 method for class 'bicop_dist'
scores(u, vinecop, cores = 1, parameters = NULL, ...)

## S3 method for class 'bicop_dist'
hessian(u, vinecop, cores = 1, parameters = NULL, ...)

hbicop(
  u,
  cond_var,
  family,
  rotation,
  parameters,
  inverse = FALSE,
  var_types = c("c", "c"),
  deriv = NULL,
  cores = 1
)
```

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. For <code>scores()</code> and <code>hessian()</code> , optional observation-specific parameters override those stored in <code>vinecop</code> : a vector is accepted for one-parameter families; otherwise, use a matrix with one row per observation and one column per parameter. Parameters are not recycled.
var_types	variable types, a length 2 vector; e.g., <code>c("c", "c")</code> for both continuous (default), or <code>c("c", "d")</code> for first variable continuous and second discrete.
log	whether to return the log-density or a derivative of the log-density.
deriv	NULL for ordinary evaluation, or a character vector of length one or two specifying a first- or second-order partial derivative. Each component is one of "u1", "u2", or "par<k>"; "par" is an alias for "par1".
cores	number of cores used when evaluating observation-specific parameters.
n	number of observations. If <code>'length(n) > 1'</code> , the length is taken to be the number required.
qrng	if TRUE, generates quasi-random numbers using the bivariate Generalized Halton sequence (default <code>qrng = FALSE</code>).
vinecop	a <code>bicop_dist</code> object for <code>scores()</code> and <code>hessian()</code> .
...	unused.
cond_var	either 1 or 2; <code>cond_var = 1</code> conditions on the first variable, <code>cond_var = 2</code> 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`.

Derivatives:

Setting `deriv` evaluates a selected first- or second-order derivative of the density, log-density, or h-function. For example, `deriv = "u1"` differentiates with respect to the first argument and `deriv = c("u1", "par2")` evaluates the corresponding mixed second derivative. Derivative order is immaterial. Parameter derivatives use the natural parameters of the (possibly rotated) copula.

Derivatives are available only for continuous parametric copulas. The backend uses analytic formulas where available and finite-difference fallbacks where needed. Derivatives of inverse h-functions are not available.

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 or log-density, and `pbicop()` gives the distribution function. `rbicop()` generates random deviates, and `hbicop()` gives the h-functions (and their inverses). If `deriv` is set, `dbicop()` and `hbicop()` return the selected observation-wise derivative. `scores()` gives the observation-wise score matrix and `hessian()` gives the average Hessian matrix for a `bicop_dist` object.

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. For vectorized simulation, parameters must have one row per generated observation and one column per family parameter.

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)

## derivatives
dbicop(c(0.2, 0.4), joe_cop, deriv = "u1")
```

```

dbicop(c(0.2, 0.4), joe_cop, log = TRUE, deriv = "par1")
hbicop(c(0.2, 0.4), 1, joe_cop, deriv = c("u1", "par1"))

## 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_set = "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)
```

kde1d_family

Define a kde1d margin family

Description

The arguments configure `kde1d::kde1d()` and travel with the family object, rather than being interpreted by `vine()`.

Usage

```
kde1d_family(xmin = NaN, xmax = NaN, mult = 1, bw = NA, deg = 2)
```

Arguments

xmin, xmax, mult, bw, deg
arguments passed to `kde1d::kde1d()`.

Value

A margin-family specification supporting all rvinecopulib variable types.

margin_dist	<i>Create a fitted or fixed custom margin</i>
-------------	---

Description

`margin_dist()` creates a fitted or fixed margin from evaluation functions and explicit metadata. The functions must each accept one vector argument and return a numeric vector of the same length.

Usage

```
margin_dist(
  d,
  p,
  q,
  family = "custom",
  type = "c",
  support = c(-Inf, Inf),
  npars = 0,
  loglik = NA_real_
)
```

Arguments

d	density or probability mass function.
p	distribution function.
q	quantile function.
family	descriptive family name.
type	margin type: "c", "d", or "zi".
support	mathematical support as its lower and upper bounds.
npars	effective number of parameters.
loglik	maximized marginal log-likelihood, or NA for a fixed margin.

Value

A fitted margin implementing the [margin protocol](#).

Examples

```
normal_margin <- margin_dist(
  d = function(x) dnorm(x, mean = 1, sd = 2),
  p = function(x) pnorm(x, mean = 1, sd = 2),
  q = function(p) qnorm(p, mean = 1, sd = 2),
  family = "norm",
  support = c(-Inf, Inf),
  npars = 2
)
dmargin(1, normal_margin)
pmargin(1, normal_margin)
qmargin(0.5, normal_margin)
```

margin_family	<i>Define a custom margin family</i>
---------------	--------------------------------------

Description

margin_family() defines a candidate family for [vine\(\)](#). Its fitting function must accept x, weights, and type, and return an object implementing the [fitted-margin protocol](#). Additional named fitting arguments can be stored in fit_args.

Usage

```
margin_family(fit, family_name = "custom", types = "c", fit_args = list())
```

Arguments

fit	function accepting x, weights, and type.
family_name	descriptive family name.
types	supported variable types.
fit_args	named list of additional arguments passed to fit.

Value

A margin-family specification.

Examples

```
normal_family <- margin_family(
  fit = function(x, weights, type) {
    if (length(weights)) {
      location <- weighted.mean(x, weights)
      scale <- sqrt(weighted.mean((x - location)^2, weights))
    } else {
      location <- mean(x)
      scale <- sd(x)
    }
    margin_dist(
      d = function(y) dnorm(y, location, scale),
      p = function(y) pnorm(y, location, scale),
      q = function(p) qnorm(p, location, scale),
      family = "normal",
      type = type,
      npars = 2,
      loglik = sum(dnorm(x, location, scale, log = TRUE))
    )
  },
  family_name = "normal"
)
```

margin_family_protocol

Margin-family fitting protocol

Description

Margin families describe how a candidate is fitted. Implementations provide `fit_margin()` and `margin_info()` methods. Their `margin_info()` result must contain `family_name` and `types`. Every fitter receives the observations, observation weights, and requested variable type. It must either use the weights or reject them.

Usage

```
fit_margin(family, x, weights = numeric(), type = "c")
```

Arguments

<code>family</code>	a margin-family specification.
<code>x</code>	observed values.
<code>weights</code>	observation weights, or <code>numeric()</code> .
<code>type</code>	requested variable type.

Value

`fit_margin()` returns a fitted margin.

margin_protocol	<i>Fitted marginal distribution protocol</i>
-----------------	--

Description

Fitted margins provide distribution evaluation and model metadata through a small set of S3 generics. Methods for `dmargin()`, `pmargin()`, and `qmargin()` dispatch on `margin`, their second argument. `margin_info()` dispatches on its only argument.

Usage

```
dmargin(x, margin)
pmargin(x, margin)
qmargin(p, margin)
margin_info(object)
```

Arguments

<code>x</code>	vector of evaluation points.
<code>margin</code>	a fitted marginal distribution object.
<code>p</code>	vector of probabilities.
<code>object</code>	a fitted margin or margin-family specification.

Details

A fitted margin class must implement all four generics documented here. Evaluation methods must be vectorized. `margin_info()` returns a list with entries `family_name`, `type`, `support`, `npars`, and `loglik`. `type` is "c" for a continuous distribution, "d" for an integer-supported distribution, or "zi" for a continuous distribution with an atom at zero. `support` contains the mathematical lower and upper bounds. `npars` must be non-negative or NA when unavailable; `loglik` may be NA for a fixed margin.

The type determines left limits throughout `rvinecopulib`. They are $F(x)$ for continuous margins, $F(x - 1)$ for integer-supported margins, and $F(0) - P(X = 0)$ at zero for zero-inflated margins. Consequently, `dmargin(0, margin)` must return the atom's mass for a zero-inflated margin.

Value

`dmargin()`, `pmargin()`, and `qmargin()` return numeric vectors. `margin_info()` returns a list of model information.

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

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

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 (<https://arxiv.org/pdf/1801.09739>)

Examples

```
u <- sapply(1:5, function(i) runif(50))
fit <- vinecop(u, family_set = "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	<i>Exploratory pairs plot for copula data</i>
-------------------	---

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)
```

parameter_uncertainty	<i>Parameter uncertainty</i>
-----------------------	------------------------------

Description

Covariance matrices and Wald confidence intervals for the estimated pair-copula parameters.

Usage

```
## S3 method for class 'vinecop_dist'
vcov(object, newdata = NULL, step_wise = TRUE, cores = 1, ...)

## S3 method for class 'bicop_dist'
vcov(object, newdata = NULL, step_wise = TRUE, cores = 1, ...)

## S3 method for class 'vinecop_dist'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'bicop_dist'
confint(object, parm, level = 0.95, ...)
```

Arguments

object	an object of class "bicop" or "vinecop", or of class "bicop_dist" or "vinecop_dist" together with newdata.
newdata	copula data to evaluate the derivatives at. Defaults to the data stored in object, which requires keep_data = TRUE at fitting time.
step_wise	if TRUE (default), derivatives of the step-wise estimating equations; if FALSE, of the joint log-likelihood. See scores() .
cores	number of cores to use; if larger than one, computations are done in parallel on cores batches.
...	passed on to vcov().
parm	a specification of which parameters to give intervals for: a vector of names or of positions. Defaults to all of them.
level	the confidence level.

Details

The estimator solves $\bar{s}(\hat{\theta}) = 0$ with $\bar{s}(\theta) = n^{-1} \sum_i s_i(\theta)$, so it is an M-estimator. Writing

$$A = -\partial \bar{s}(\theta) / \partial \theta^\top, \quad B = n^{-1} \sum_i s_i s_i^\top,$$

vcov() returns the sandwich $A^{-1} B A^{-\top} / n$ of White (1982).

Which objective is differentiated:

step_wise selects the estimating equation, and it changes the nature of A .

With step_wise = FALSE, $\bar{s}$ is the gradient of the joint log-likelihood and A is minus its averaged Hessian: a symmetric matrix, and the classical setting for both formulas.

With step_wise = TRUE (the default), the pseudo-observations entering each pair copula are held fixed. This is the estimating equation that sequential, tree-by-tree estimation actually solves, but it is not the gradient of any objective: A is a Jacobian rather than a Hessian, and it is block triangular rather than symmetric, which is why the sandwich is written with $A^{-\top}$ on the right.

Restrictions:

No derivatives are implemented for nonparametric pair copulas, so a model containing one is refused. For models with discrete variables the backend differentiates by central finite differences rather than in closed form; the result is correct but carries the accuracy of a finite-difference approximation.

Why there is no method for vine distributions:

These functions treat the copula data as observed, so they quantify uncertainty in the pair-copula parameters *given* the margins. A model fitted by [vine\(\)](#) estimates its margins too, and the second stage inherits the first stage's sampling variability: intervals that ignore it are too short. In a small experiment (three-dimensional Gaussian D-vine, equicorrelation 0.5, $n = 1000$, 1000 replications) nominal 95% intervals attained 0.956 when the true margins were used and 0.929 when they were estimated by ranks.

Correcting for it needs the influence function of the marginal estimator, and [fit_margin\(\)](#) is a user-supplied black box, so there is nothing to differentiate. We therefore provide no method for

"vine" objects rather than a silently anticonservative one. To quantify uncertainty for a complete vine distribution, resample:

```
boot <- replicate(B, {
  idx <- sample(nrow(x), replace = TRUE)
  fit <- vine(x[idx, ], margins_controls = mc, copula_controls = cc)
  unlist(get_all_parameters(fit$copula))
})
apply(boot, 1, quantile, c(0.025, 0.975))
```

Value

`vcov()` returns a covariance matrix with one row and column per estimated pair-copula parameter, ordered by tree, then edge, then parameter within the pair copula, with dimnames giving that position. `confint()` returns a matrix with columns for the lower and upper endpoints.

References

White H (1982). "Maximum Likelihood Estimation of Misspecified Models." *Econometrica*, **50**(1), 1–25. doi:[10.2307/1912526](https://doi.org/10.2307/1912526)

Stoeber J, Schepsmeier U (2013). "Estimating Standard Errors in Regular Vine Copula Models." *Computational Statistics*, **28**(6), 2679–2707. doi:[10.1007/s0018001304238](https://doi.org/10.1007/s0018001304238)

See Also

[scores\(\)](#), [hessian\(\)](#), [vinecop\(\)](#), [vine\(\)](#)

Examples

```
# bivariate copula -----
u <- rbicop(500, "gaussian", 0, 0.5)
fit <- bicop(u, family_set = "parametric", keep_data = TRUE)
sqrt(diag(vcov(fit)))
confint(fit)

# vine copula -----
u <- pseudo_obs(matrix(rnorm(500 * 3), 500, 3))
fit <- vinecop(u, family_set = "parametric", keep_data = TRUE)
vcov(fit)
confint(fit, level = 0.9)

# derivatives of the joint log-likelihood instead of the step-wise ones
sqrt(diag(vcov(fit, step_wise = FALSE)))
```

par_to_ktau	<i>Conversion between Kendall's tau and parameters</i>
-------------	--

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 <code>bicop_dist()</code>) or a <code>bicop_dist</code> 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 <code>bicop_dist</code> 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	<i>Plotting tools for bicop_dist and bicop objects</i>
-----------------	--

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	<i>Plotting vinecop_dist and vinecop objects.</i>
-------------------	---

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 $1.25/\text{length}(\text{tree}) * (\text{d} - \min(\text{tree}))$ 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_set = "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 <code>rank()</code> (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 $x_i = (x_{i1}, \dots, x_{id})$, $i \in \{1, \dots, n\}$ of a random vector X , the pseudo-observations are defined via $u_{ij} = r_{ij}/(n+1)$ for $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, d\}$, where r_{ij} denotes the rank of x_{ij} among all x_{kj} , $k \in \{1, \dots, n\}$.

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 $1 - \text{pseudo_obs}()$.

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

*Rosenblatt and inverse Rosenblatt transforms***Description**

The Rosenblatt transform maps observations from a model to independent uniform variates. The inverse transform evaluates conditional quantiles and maps independent uniforms to draws from the model; see *Details*.

Usage

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

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

Arguments

<code>x</code>	matrix of evaluation points; must be in $(0, 1)^d$ for copula models.
<code>model</code>	a model object; classes currently supported are <code>bicop_dist()</code> , <code>vinecop_dist()</code> , and <code>vine_dist()</code> .
<code>cores</code>	if >1, computation is parallelized over cores batches (rows of <code>u</code>).
<code>randomize_discrete</code>	Whether to randomize the transform for discrete variables; see <i>Details</i> .
<code>conditioning_set</code>	optional variable indices or names that define the conditioning variables. The transform uses an admissible sampling order whose tail contains exactly this set. The model is not modified. If <code>NULL</code> , the current model order is used. An error is thrown if the requested set cannot form an admissible tail.
<code>u</code>	matrix of evaluation points; must be in $(0, 1)^d$.

Details

Let $(s_1, \dots, s_d)$ be the sampling order of a random vector $V = (V_1, \dots, V_d)$ with distribution F . For continuous variables, the Rosenblatt transform $Z = T(V)$ is defined by

$$Z_{s_1} = F_{s_1}(V_{s_1}), \quad Z_{s_j} = F_{s_j|s_1, \dots, s_{j-1}}(V_{s_j} | V_{s_1}, \dots, V_{s_{j-1}}), \quad j = 2, \dots, d.$$

If the model is correct, the components of Z are independent standard uniforms. The result is stored in the original variable columns; the sampling order determines only the sequence of conditional distributions.

The inverse transform applies the corresponding conditional quantiles:

$$V_{s_1} = F_{s_1}^{-1}(Z_{s_1}), \quad V_{s_j} = F_{s_j|s_1, \dots, s_{j-1}}^{-1}(Z_{s_j} | V_{s_1}, \dots, V_{s_{j-1}}), \quad j = 2, \dots, d.$$

Thus $T^{-1}(Z)$ has distribution F when Z contains independent standard uniforms.

If a variable has atoms, its conditional cdf jumps at the observation. Let G_j denote the conditional cdf of V_{s_j} given the preceding variables in the sampling order. Following Brockwell (10.1016/j.spl.2007.02.008), `rosenblatt()` returns

$$Z_{s_j} = W_j G_j(V_{s_j}) + (1 - W_j) G_j(V_{s_j}^-),$$

where $G_j(V_{s_j}^-)$ is the left limit and the W_j are independent standard uniforms. This randomization is used by default and yields uniform components under the fitted model. Set `randomize_discrete = FALSE` to return the upper endpoint $G_j(V_{s_j})$ instead.

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)
```

Description

rvinecopulib provides high-performance tools for constructing, fitting, selecting, simulating, and visualizing bivariate and vine copula models. It also fits complete multivariate distributions by combining marginal models with a vine copula. Continuous, integer-valued discrete, mixed, and zero-inflated variables are supported.

Details

The package is the R interface to the header-only 'rvinecopulib' C++ library, which is bundled with rvinecopulib. Users do not need to install the C++ library separately. 'rvinecopulib' is licensed under the MIT License and rvinecopulib under the GNU GPL version 3.

Main capabilities

rvinecopulib provides:

- parametric, nonparametric, rotated, and extreme-value pair copulas;
- automatic pair-copula family, vine structure, truncation, and threshold selection;
- complete multivariate distributions with nonparametric, parametric, or custom margins;
- continuous, integer-valued discrete, mixed, and zero-inflated data;
- observation weights and parallel fitting;
- conditional simulation and Rosenblatt transforms;
- likelihood scores, Hessians, and observation-specific parameters;
- custom marginal families and custom tree-selection criteria.

Modeling interfaces

The framework is exposed at three levels. `vine()` models observations on their original scale by fitting the marginal distributions and dependence model together. `vinecop()` models uniform pseudo-observations when only the dependence model is required. `bicop()` provides the corresponding two-variable workflow.

The constructors `vine_dist()`, `vinecop_dist()`, and `bicop_dist()` create models from explicitly specified components.

Learning more

Start with `vignette("getting-started")`. Dedicated articles cover bivariate copulas, vine structures and selection, marginal models, discrete data, conditional simulation, and likelihood derivatives. The [package website](#) contains the rendered articles and complete function reference.

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

```
## complete distribution on the original data scale
x <- data.frame(a = rnorm(50), b = rexp(50), n = rpois(50, 2))
fit <- vine(
  x,
  var_types = c("c", "c", "d"),
  copula_controls = list(family_set = "onepar")
)
rvine(3, fit)

## dependence model on the copula scale
u <- pseudo_obs(as.matrix(USArrests))
copula_fit <- vinecop(u, family_set = "onepar")
summary(copula_fit)

## bivariate model
uv <- rbicop(100, "gaussian", 0, 0.5)
bicop(uv, family_set = "par")
```

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-vine 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 <i>Details</i> .

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 $(M[d + 1 - e, e], M[t, e]; M[t - 1, e], \dots, M[1, e])$). 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 of 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 $t = 1, \dots, d - 2$ and $e = 1, \dots, d - t$ there must exist an index $j > d$, such that $(M[t, e], \{M[1, e], \dots, M[t - 1, e]\})$ equals either $(M[d + 1 - j, j], \{M[1, j], \dots, M[t - 1, j]\})$ or $(M[t - 1, j], \{M[d + 1 - j, j], M[1, j], \dots, M[t - 2, j]\})$.

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

<code>d</code>	the number of variables
<code>natural_order</code>	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)
```

stats_margin

Create a fixed margin from a stats distribution

Description

`stats_margin()` adapts the univariate distributions documented in [stats::Distributions](#) to the fitted-margin protocol. Distribution parameters are supplied through `...`; the resulting margin reports the distribution's parameter count but has no fitted log-likelihood.

Usage

```
stats_margin(family, ...)
```

Arguments

`family` distribution suffix such as "norm", "pois", or "binom".

`...` parameters passed to the distribution's density, CDF, and quantile functions.

Value

A fixed margin implementing the [margin protocol](#).

Examples

```
normal <- stats_margin("norm", mean = 1, sd = 2)
poisson <- stats_margin("pois", lambda = 3)
margin_info(normal)
margin_info(poisson)
```

truncate_model	<i>Truncate a vine copula model</i>
----------------	-------------------------------------

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)

```

univariateML_family	<i>Define a univariateML margin family</i>
---------------------	--

Description

univariateML_family() adapts one of univariateML's available models to the margin-family protocol.

Usage

```
univariateML_family(family)
```

Arguments

family a univariateML family name.

Value

A parametric margin-family specification.

vine	<i>Vine copula models</i>
------	---------------------------

Description

Automated fitting or creation of custom vine copula models

Usage

```

vine(
  data,
  margins_controls = list(),
  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", conditioning_set
    = NULL),
  weights = numeric(),
  keep_data = FALSE,
  cores = 1,
  var_types = NULL
)

vine_dist(margins, pair_copulas, structure)

```

Arguments

<code>data</code>	a matrix or data.frame. Discrete variables have to be declared as <code>ordered()</code> .
<code>margins_controls</code>	a list with arguments to be passed to marginal fitting. The supported entries are • <code>family_set</code> candidate families used for every variable, or a list with one entry per variable. The default "kde1d" preserves nonparametric margin fitting. Other character names refer to univariateML families; configured built-in and custom candidates are created with <code>kde1d_family()</code>, <code>univariateML_family()</code>, and <code>margin_family()</code>. • <code>selcrit</code> selection criterion, one of "loglik", "aic", or "bic". • <code>cores</code> number of cores used for marginal fitting. NULL inherits the top-level <code>cores</code> argument. Use one core for stochastic custom fitters when results must be invariant to the number of cores.
<code>copula_controls</code>	a list with arguments to be passed to <code>vinecop()</code> .
<code>weights</code>	optional numeric vector of weights for each observation, used for both marginal and copula estimation. Every margin family receives these weights and is responsible for using or explicitly rejecting them.
<code>keep_data</code>	whether the original data should be stored; if you want to store the pseudo-observations used for fitting the copula, use the <code>copula_controls</code> argument.
<code>cores</code>	the number of cores to use for parallel computations. Unless overridden by <code>margins_controls\$cores</code> , margins are fitted in forked processes when <code>cores > 1</code> on non-Windows systems and serially on Windows. Stochastic custom margin fits may depend on the number of forked processes.
<code>var_types</code>	optional variable types, one for each variable after factor expansion: "c" for continuous, "d" for integer-valued discrete, or "zi" for continuous with an atom at zero. Types are inferred from <code>ordered()</code> and <code>zero_inflated()</code> columns when omitted.

margins	a list containing one marginal distribution per variable. Each margin can be a margin_dist() object, another fitted object implementing the margin protocol , a kde1d object, or a stats_margin() object. Legacy <code>list(distr = ...)</code> specifications are converted through stats_margin() . If margins has length one, it is recycled for every component.
pair_copulas	A nested list of 'bicop_dist' objects, where <code>pair_copulas[[t]][[e]]</code> corresponds to the pair-copula at edge <code>e</code> in tree <code>t</code> .
structure	an <code>rvine_structure</code> 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 <code>length(pair_copulas[[1]]) + 1</code> .

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. Margin-specific fitting options are properties of the family specification, not controls interpreted by `vine()`. For example, use `kde1d_family(xmin = 0, mult = 2)` in `family_set`. `copula_controls` is a list with the same parameters as [vinecop\(\)](#) (except for data).

A character `margins_controls$family_set` is used for every variable. Its entries can be "kde1d" or names of models available in `univariateML`. A list supplies one candidate set per variable, for example `list(c("norm", "cauchy"), c("pois", "geom"))`. Each entry may itself be a character vector, a margin-family object, or a list combining both. An unnamed list is matched by position; a named list must contain every expanded variable name exactly once and is reordered to match the data.

The aliases "par" and "parametric" expand to all `univariateML` families, "nonpar" and "nonparametric" select "kde1d", and "all" combines both. Parametric names and aliases require the suggested `univariateML` package; the default "kde1d", [kde1d_family\(\)](#), and custom [margin_family\(\)](#) objects do not. Identical candidate specifications are fitted only once.

Candidates incompatible with a variable's type are removed before fitting. `rvinecopulib` fits every remaining candidate and selects the first minimum of negative log-likelihood, AIC, or BIC according to `selcrit`. Competing candidates therefore need a finite `margin_info()`\$`loglik` value, and AIC or BIC additionally requires a finite `margin_info()`\$`npars`. A sole candidate without a parameter count is retained with a warning, but model AIC and BIC are then unavailable. An unsupported candidate may fail while another candidate succeeds; such failures and all candidate warnings are reported. Protocol violations always stop fitting. The built-in `univariateML` family explicitly rejects observation weights.

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 controls used to fit and select the marginal families.
- `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 fitted objects implementing the [margin protocol](#).

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(stats_margin("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)

# parametric margin selection (requires the suggested univariateML package)
if (requireNamespace("univariateML", quietly = TRUE)) {
  fit_par_margins <- vine(
    x,
    margins_controls = list(family_set = c("norm", "cauchy"))
  )
}

## 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",
  conditioning_set = NULL
)
```

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 <i>Details</i> .
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 <code>rvine_structure</code> object, namely a compressed representation of the vine structure, or an object that can be coerced into one (see <code>rvine_structure()</code> and <code>as_rvine_structure()</code>). The dimension must be <code>length(pair_copulas[[1]]) + 1</code> ; <code>structure = NA</code> performs automatic selection based on Dissman's algorithm. See <i>Details</i> for partial selection of the structure.
par_method	the estimation method for parametric models, either <code>"mle"</code> for maximum likelihood or <code>"itau"</code> for inversion of Kendall's tau (only available for one-parameter families and <code>"t"</code>).
nonpar_method	the estimation method for nonparametric models, either <code>"constant"</code> for the standard transformation estimator, or <code>"linear"/"quadratic"</code> 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 <code>"loglik"</code> , <code>"aic"</code> , <code>"bic"</code> , <code>"mbic"</code> . For <code>vinecop()</code> there is the additional option <code>"mbicv"</code> .
weights	optional vector of weights for each observation.
psi0	prior probability of a non-independence copula (only used for <code>selcrit = "mbic"</code> and <code>selcrit = "mbicv"</code>).
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; <code>Inf</code> means no truncation, <code>NA</code> indicates that the truncation level should be selected automatically by <code>mBICV()</code> .
tree_crit	the criterion for tree selection, one of <code>"tau"</code> , <code>"rho"</code> , <code>"hoeffd"</code> , <code>"mcor"</code> , <code>"cxi"</code> , or <code>"joe"</code> for Kendall's τ , Spearman's ρ , Hoeffding's D , maximum correlation, symmetrized Chatterjee's ξ , or logarithm of the partial correlation, respectively. Alternatively, a function with arguments <code>data</code> and <code>weights</code> may be supplied. <code>data</code> is a two-column matrix of pair-copula pseudo-observations. <code>weights</code> contains the corresponding observation weights, standardized by the backend to sum to the original number of observations, or <code>numeric(0)</code> when no weights were supplied. Rows containing missing values or zero weights are removed before the function is called. The function must return one finite numeric value; its absolute value is used as the edge strength. Custom functions always run serially on the calling thread; pair-copula fitting may still use cores threads.
threshold	for thresholded vine copulas; <code>NA</code> indicates that the threshold should be selected automatically by <code>mBICV()</code> .
keep_data	whether the data should be stored (necessary for using <code>fitted()</code>).
vinecop_object	a <code>vinecop</code> 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_crit). "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).

conditioning_set variable indices or names to place at the end of the vine order. The resulting model can be sampled conditionally with `rvinecop()` by supplying `u_cond`. Conditioning-aware selection supports fixed or automatically selected truncation levels and requires an MST tree algorithm ("mst_prim" or "mst_kruskal").

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 $n \times d$ block contains realizations of $F_{X_j}(X_j)$. The second $n \times d$ 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_crit` argument, which can be set to "tau", "rho", "hoeffd", "mcor", "cxi", or "joe", or to a custom function. 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 `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 BIC from one-parameter 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_set = "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_set = "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	<i>Vine copula models</i>
--------------	---------------------------

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 <code>'bicop_dist()'</code> objects, where <code>pair_copulas[[t]][[e]]</code> corresponds to the pair-copula at edge <code>e</code> in tree <code>t</code> .
structure	an <code>rvine_structure</code> object, namely a compressed representation of the vine structure, or an object that can be coerced into one (see <code>rvine_structure()</code> and <code>as_rvine_structure()</code>). The dimension must be <code>length(pair_copulas[[1]]) + 1</code> ; <code>structure = NA</code> performs automatic selection based on Dissman's algorithm. See <i>Details</i> for partial selection of the structure.
var_types	variable types, a length <code>d</code> vector; e.g., <code>c("c", "c")</code> for two continuous variables, or <code>c("c", "d")</code> 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,
  keep_all = FALSE,
  parameters = NULL,
  log = FALSE
)

scores(u, vinecop, ...)

## S3 method for class 'vinecop_dist'
scores(u, vinecop, step_wise = TRUE, cores = 1, parameters = NULL, ...)
```

```

hessian(u, vinecop, ...)

## S3 method for class 'vinecop_dist'
hessian(u, vinecop, step_wise = TRUE, cores = 1, parameters = NULL, ...)

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

rvinecop(
  n,
  vinecop,
  qrng = FALSE,
  cores = 1,
  u_cond = NULL,
  conditioning_set = NULL
)

```

Arguments

<code>u</code>	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 <i>Details</i> .
<code>vinecop</code>	an object of class "vinecop_dist".
<code>cores</code>	number of cores to use; if larger than one, computations are done in parallel on cores batches .
<code>keep_all</code>	if TRUE, <code>dvinecop()</code> returns additional intermediate quantities computed during density evaluation.
<code>parameters</code>	optional observation-specific parameters for <code>dvinecop()</code> , <code>scores()</code> , and <code>hessian()</code> . For a model with one parameter, this may be a vector with one entry per observation. Otherwise, it must be a matrix with one row per observation and one column per model parameter. For vine copulas, columns follow the (tree, edge, parameter) order of <code>scores()</code> . Parameters are not recycled. Only continuous parametric models are supported.
<code>log</code>	if TRUE, <code>dvinecop()</code> returns the log-density instead of the density. A vine density is a product of one factor per edge, so it underflows to 0 in high dimensions or under strong dependence while its logarithm is still an ordinary double; <code>log = TRUE</code> accumulates in log space and is the accurate way to obtain it. Ignored when <code>keep_all = TRUE</code> , which reports both.
<code>...</code>	unused.
<code>step_wise</code>	if FALSE, the score/Hessian is computed for the full likelihood; if TRUE, gradients are computed per pair-copula as in step-wise estimation.
<code>n_mc</code>	number of samples used for quasi Monte Carlo integration.
<code>n</code>	number of observations.
<code>qrng</code>	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 <code>qrng = FALSE</code>).

<code>u_cond</code>	optional conditioning values for <code>rvinecop()</code> . A vector or one-row matrix is repeated n times; alternatively, supply an n -row matrix for observation-specific conditioning values. The first block holds the copula-scale values $F(x)$. The expanded layout appends one left-limit column $F(x^-)$ per conditioning variable in the same order; columns for continuous variables equal their value columns. In the compact layout, the redundant columns for continuous variables are omitted. If <code>NULL</code> , <code>rvinecop()</code> performs unconditional simulation.
<code>conditioning_set</code>	variable indices or names corresponding to the first block of <code>u_cond</code> . When <code>NULL</code> , the columns of <code>u_cond</code> correspond to the last variables of the current vine order. When supplied, the model is transiently reoriented and is not modified.

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 $n \times d$ block contains realizations of $F_{X_j}(X_j)$. The second $n \times d$ 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, `rvinecop()` generates unconditional or conditional random deviates, `scores()` gives the observation-wise score matrix, and `hessian()` gives the average Hessian matrix.

If `keep_all = TRUE`, `dvinecop()` returns a list with entries `pdf`, `logpdf`, `pdf_edges`, `hfunc1`, `hfunc2`, `hfunc1_sub`, and `hfunc2_sub`. The `_sub` entries contain h-functions evaluated at left-sided limits when the model contains discrete variables; they are empty for fully continuous models.

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_set = "clayton")

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

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

# evaluate derivatives of the log-likelihood
scores(u, vc)
hessian(u, vc)

# derivatives can also be computed for the full likelihood
scores(u, vc, step_wise = FALSE)
hessian(u, vc, step_wise = FALSE)

## 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))

## Conditional simulation
vc_cond <- vinecop(
  u,
  family_set = "gaussian",
  conditioning_set = c(2, 4)
)
u_cond <- c(0.25, 0.75)
uc <- rvinecop(
  100,
  vc_cond,
  u_cond = u_cond,
  conditioning_set = c(2, 4)
)
stopifnot(
  isTRUE(all.equal(uc[, 2], rep(0.25, 100))),
  isTRUE(all.equal(uc[, 4], rep(0.75, 100)))
)

```

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 $n \times d$ block contains realizations of $F_{X_j}(X_j)$. The second $n \times d$ 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_set = "par", keep_data = TRUE)
all.equal(predict(fit, u), fitted(fit), check.environment = FALSE)
```

vine_distributions	<i>Vine based distributions</i>
--------------------	---------------------------------

Description

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

Usage

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

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

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

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 .
log	if TRUE, dvine() returns the log-density instead of the density. The density is accumulated in log space either way and only exponentiated at the end; log = TRUE skips that last step. The joint density is a product over the margins and the vine edges, so it underflows to 0 in high dimensions or under strong dependence while its logarithm is still an ordinary double.
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).
x_cond	optional conditioning values for rvine() on the original data scale. A vector or one-row object is repeated n times; alternatively, supply an n-row matrix or data frame for observation-specific conditioning values. If NULL, rvine() performs unconditional simulation.
conditioning_set	variable indices or names corresponding to the columns of x_cond. When NULL, the columns correspond to the last variables of the current copula order. Discrete left limits are computed internally from the fitted margins.

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 [dvinetop\(\)](#), [pvinecop\(\)](#) and [rvinecop\(\)](#) for [vinecop](#) objects. Margins are evaluated through [dmargin\(\)](#), [pmargin\(\)](#), and [qmargin\(\)](#). Methods are provided for margins fitted by [vine\(\)](#) and for the fixed [stats::Distributions](#) specifications accepted by [vine_dist\(\)](#).

Value

[dvinetop\(\)](#) gives the density, [pvinecop\(\)](#) gives the distribution function, and [rvinecop\(\)](#) generates unconditional or conditional 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 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(stats_margin("norm")), pcs, mat)

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

# evaluate the density and cdf
dvinetop(x[1, ], vc)
pvinecop(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)
```

zero_inflated	<i>Declare zero-inflated data</i>
---------------	-----------------------------------

Description

Marks a numeric vector as continuous with an atom at zero. The class is preserved when stored in or subset from a data frame and is detected by [vine\(\)](#). Alternatively, use var_types = "zi".

Usage

```
zero_inflated(x)
```

Arguments

x	a numeric vector.
---	-------------------

Value

x with an additional zero_inflated class.

Index

- * **package**
 - rvinecopulib, 33
- * **plot**
 - plot.bicop_dist, 26
 - plot.vinecop_dist, 28
- as.bicop, 3
- as_margin, 3
- as_rvine_matrix (as_rvine_structure), 4
- as_rvine_matrix(), 35, 36
- as_rvine_structure, 4
- as_rvine_structure(), 35, 36, 42, 45, 48
- bicop, 6, 9, 12, 17
- bicop(), 8, 13, 33, 44
- bicop_dependence, 8
- bicop_dist, 9, 9, 13, 26
- bicop_dist(), 8, 10, 13, 26, 27, 33, 48
- bicop_distributions, 11
- bicop_predict_and_fitted, 14
- blomqvist_beta (bicop_dependence), 8
- confint.bicop_dist
 - (parameter_uncertainty), 23
- confint.vinecop_dist
 - (parameter_uncertainty), 23
- contour.bicop, 29
- contour.bicop (plot.bicop_dist), 26
- contour.bicop(), 8, 10
- contour.bicop_dist (plot.bicop_dist), 26
- contour.vinecop (plot.vinecop_dist), 28
- contour.vinecop(), 47, 51
- contour.vinecop_dist
 - (plot.vinecop_dist), 28
- contour.vinecop_dist(), 49
- cvine_structure (rvine_structure), 34
- dbicop (bicop_distributions), 11
- dbicop(), 8, 10, 14
- dbicop_dist (bicop_distributions), 11
- dmargin (margin_protocol), 21
- dmargin(), 55
- dvine (vine_distributions), 54
- dvine(), 56
- dvine_dist (vine_distributions), 54
- dvine_structure (rvine_structure), 34
- dvinecop (vinecop_distributions), 49
- dvinecop(), 47, 49, 53, 55
- dvinecop_dist (vinecop_distributions), 49
- emp_cdf, 15
- fit_margin (margin_family_protocol), 20
- fit_margin(), 24
- fitted(), 7, 45
- fitted-margin protocol, 19
- fitted.bicop
 - (bicop_predict_and_fitted), 14
- fitted.vine (vine_predict_and_fitted), 55
- fitted.vinecop
 - (vinecop_predict_and_fitted), 53
- get_all_families (getters), 16
- get_all_ktaus (getters), 16
- get_all_pair_copulas (getters), 16
- get_all_parameters (getters), 16
- get_family (getters), 16
- get_family(), 17
- get_ktau (getters), 16
- get_ktau(), 17
- get_matrix, 16
- get_matrix (getters), 16
- get_pair_copula (getters), 16
- get_pair_copula(), 17
- get_parameters (getters), 16
- get_parameters(), 17
- get_structure, 16

get_structure (getters), 16
 getters, 16
 graphics::contour(), 27

 hbicop (bicop_distributions), 11
 hbicop(), 8, 10, 14
 hbicop_dist (bicop_distributions), 11
 hessian (vinecop_distributions), 49
 hessian(), 25
 hessian.bicop_dist
 (bicop_distributions), 11

 inverse_rosenblatt (rosenblatt), 31
 is.rvine_matrix (rvine_structure), 34
 is.rvine_structure (rvine_structure), 34

 kde1d::kde1d(), 17, 18
 kde1d_family, 17
 kde1d_family(), 41, 42
 ktau_to_par (par_to_ktau), 26

 lattice::wireframe(), 27

 margin_protocol, 19, 38, 42, 43
 margin_dist, 18
 margin_dist(), 42
 margin_family, 19
 margin_family(), 41, 42
 margin_family_protocol, 20
 margin_info (margin_protocol), 21
 margin_protocol, 21
 mBICV, 22
 mBICV(), 7, 45

 ordered(), 41

 pairs_copula_data, 23
 par_to_ktau, 26
 parameter_uncertainty, 23
 pbicop (bicop_distributions), 11
 pbicop(), 8, 10, 14
 pbicop_dist (bicop_distributions), 11
 plot.bicop, 28, 29
 plot.bicop (plot.bicop_dist), 26
 plot.bicop(), 8, 10
 plot.bicop_dist, 26
 plot.rvine_matrix
 (plot.rvine_structure), 27
 plot.rvine_matrix(), 36, 38
 plot.rvine_structure, 27

 plot.rvine_structure(), 36, 38
 plot.vinecop (plot.vinecop_dist), 28
 plot.vinecop(), 47, 51
 plot.vinecop_dist, 28
 plot.vinecop_dist(), 49
 pmargin (margin_protocol), 21
 pmargin(), 55
 predict.bicop
 (bicop_predict_and_fitted), 14
 predict.bicop_dist
 (bicop_predict_and_fitted), 14
 predict.vine (vine_predict_and_fitted),
 55
 predict.vinecop
 (vinecop_predict_and_fitted),
 53
 pseudo_obs, 30
 pvine (vine_distributions), 54
 pvine(), 56
 pvine_dist (vine_distributions), 54
 pvinecop (vinecop_distributions), 49
 pvinecop(), 47, 49, 53, 55
 pvinecop_dist (vinecop_distributions),
 49

 qmargin (margin_protocol), 21
 qmargin(), 55

 rank(), 30
 rbicop (bicop_distributions), 11
 rbicop(), 8, 10
 rbicop_dist (bicop_distributions), 11
 rosenblatt, 31
 rvine (vine_distributions), 54
 rvine_dist (vine_distributions), 54
 rvine_matrix (rvine_structure), 34
 rvine_matrix(), 4, 5, 36–38, 49
 rvine_matrix_sim (rvine_structure_sim),
 37
 rvine_matrix_sim(), 36
 rvine_structure, 16, 34
 rvine_structure(), 4, 5, 36–38, 42, 45, 48,
 49
 rvine_structure_sim, 37
 rvine_structure_sim(), 36
 rvinecop (vinecop_distributions), 49
 rvinecop(), 46, 47, 49, 55
 rvinecop_dist (vinecop_distributions),
 49

rvinecopulib, 33
rvinecopulib-package (rvinecopulib), 33

scores (vinecop_distributions), 49
scores(), 24, 25
scores.bicop_dist
 (bicop_distributions), 11
stats::Distributions, 38, 55
stats_margin, 38
stats_margin(), 3, 42

tail_dep (bicop_dependence), 8
truncate_model, 39

univariateML_family, 40
univariateML_family(), 41

vcov.bicop_dist
 (parameter_uncertainty), 23
vcov.vinecop_dist
 (parameter_uncertainty), 23
vine, 40, 55
vine(), 17, 19, 24, 25, 33, 42, 43, 55, 56
vine_dist (vine), 40
vine_dist(), 33, 42, 43, 55
vine_distributions, 54
vine_predict_and_fitted, 55
vinecop, 44, 55
vinecop(), 25, 33, 41–43, 47, 49, 51
vinecop_dist, 29, 48
vinecop_dist(), 33, 42, 47, 51
vinecop_distributions, 49
vinecop_predict_and_fitted, 52

wdm: : wdm, 46

zero_inflated, 56
zero_inflated(), 41