

Package ‘CondCopulas’

November 24, 2025

Type Package

Title Estimation and Inference for Conditional Copula Models

Version 0.2.0

Description Provides functions for the estimation of conditional copulas models, various estimators of conditional Kendall's tau (proposed in Derumigny and Fermanian (2019a, 2019b, 2020) [doi:10.1515/demo-2019-0016](https://doi.org/10.1515/demo-2019-0016)), [doi:10.1016/j.csda.2019.01.013](https://doi.org/10.1016/j.csda.2019.01.013), [doi:10.1016/j.jmva.2020.104610](https://doi.org/10.1016/j.jmva.2020.104610)), test procedures for the simplifying assumption (proposed in Derumigny and Fermanian (2017) [doi:10.1515/demo-2017-0011](https://doi.org/10.1515/demo-2017-0011)) and Derumigny, Fermanian and Min (2022) [doi:10.1002/cjs.11742](https://doi.org/10.1002/cjs.11742)), and measures of non-simplifyingness (proposed in Derumigny (2025) [doi:10.48550/arXiv.2504.07704](https://doi.org/10.48550/arXiv.2504.07704)).

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.3

Suggests MASS, knitr, rmarkdown, DiagrammeR, ggplot2, mvtnorm, testthat (>= 3.0.0)

Imports VineCopula, pbapply, glmnet, ordinalNet, tree, nnet, data.tree, statmod, wdm

VignetteBuilder knitr

BugReports <https://github.com/AlexisDerumigny/CondCopulas/issues>

URL <https://github.com/AlexisDerumigny/CondCopulas>

Config/testthat/edition 3

NeedsCompilation no

Author Alexis Derumigny [aut, cre] (ORCID: <https://orcid.org/0000-0002-6163-8097>), Jean-David Fermanian [ctb, ths] (ORCID: <https://orcid.org/0000-0001-5960-5555>), Aleksey Min [ctb] (ORCID: <https://orcid.org/0000-0001-6928-4556>), Rutger van der Spek [ctb]

Maintainer Alexis Derumigny <a.f.f.derumigny@tudelft.nl>

Repository CRAN

Date/Publication 2025-11-24 15:50:01 UTC

Contents

bCond.estParamCopula	3
bCond.pobs	4
bCond.simpA.CKT	5
bCond.simpA.param	8
bCond.treeCKT	10
CKT.estimate	12
CKT.fit.GLM	15
CKT.fit.nNets	17
CKT.fit.randomForest	18
CKT.fit.tree	21
CKT.hCV.lout	22
CKT.kendallReg.fit	25
CKT.KendallReg.LambdaCV	28
CKT.kernel	30
CKT.predict.kNN	34
CKT.predict.nNets	37
CKTmatrix.kernel	37
computeKernelMatrix	40
computeMatrixSignPairs	41
conv_treeCKT	42
datasetPairs	43
estimateCondCDF_matrix	45
estimateCondCDF_vec	46
estimateCondQuantiles	47
estimateNPCondCopula	48
estimateParCondCopula	49
measures_nonsimplifyingness_NP	51
se	53
se.estimated_CKT_kernel	53
simpA.kendallReg	55
simpA.NP	59
simpA.param	62

Index

65

bCond.estParamCopula	<i>Estimation of the conditional parameters of a parametric conditional copula with discrete conditioning events.</i>
----------------------	---

Description

By Sklar's theorem, any conditional distribution function can be written as

$$F_{1,2|A}(x_1, x_2) = c_{1,2|A}(F_{1|A}(x_1), F_{2|A}(x_2)),$$

where A is an event and $c_{1,2|A}$ is a copula depending on the event A . In this function, we assume that we have a partition A_1, \dots, A_p of the probability space, and that for each $k = 1, \dots, p$, the conditional copula is parametric according to the following model

$$c_{1,2|Ak} = c_{\theta(Ak)},$$

for some parameter $\theta(Ak)$ depending on the realized event Ak . This function uses canonical maximum likelihood to estimate $\theta(Ak)$ and the corresponding copulas $c_{1,2|Ak}$.

Usage

```
bCond.estParamCopula(U1, U2, family, partition)
```

Arguments

U1	vector of n conditional pseudo-observations of the first conditioned variable.
U2	vector of n conditional pseudo-observations of the second conditioned variable.
family	the family of conditional copulas used for each conditioning event A_k . If not of length p , it is recycled to match the number of events p .
partition	matrix of size $n * p$, where p is the number of conditioning events that are considered. <code>partition[i,j]</code> should be the indicator of whether the i -th observation belongs or not to the j -th conditioning event

Value

a list of size p containing the p conditional copulas

References

Derumigny, A., & Fermanian, J. D. (2017). About tests of the “simplifying” assumption for conditional copulas. *Dependence Modeling*, 5(1), 154-197. doi:10.1515/demo20170011

Derumigny, A., & Fermanian, J. D. (2022) Conditional empirical copula processes and generalized dependence measures *Electronic Journal of Statistics*, 16(2), 5692-5719. doi:10.1214/22EJS2075

See Also

[bCond.pobs](#) for the computation of (conditional) pseudo-observations in this framework.

[bCond.simpA.param](#) for a test of the simplifying assumption that all these conditional copulas are equal (assuming they all belong to the same parametric family). [bCond.simpA.CKT](#) for a test of the simplifying assumption that all these conditional copulas are equal, based on the equality of conditional Kendall's tau.

Examples

```
n = 800
Z = stats::runif(n = n)
CKT = 0.2 * as.numeric(Z <= 0.3) +
      0.5 * as.numeric(Z > 0.3 & Z <= 0.5) +
      - 0.8 * as.numeric(Z > 0.5)
simCopula = VineCopula::BiCopSim(N = n,
  par = VineCopula::BiCopTau2Par(CKT, family = 1), family = 1)
X1 = simCopula[,1]
X2 = simCopula[,2]
partition = cbind(Z <= 0.3, Z > 0.3 & Z <= 0.5, Z > 0.5)
condPseudoObs = bCond.pobs(X = cbind(X1, X2), partition = partition)

estimatedCondCopulas = bCond.estParamCopula(
  U1 = condPseudoObs[,1], U2 = condPseudoObs[,2],
  family = 1, partition = partition)
print(estimatedCondCopulas)
# Comparison with the true conditional parameters: 0.2, 0.5, -0.8.
```

bCond.pobs

Computing the pseudo-observations in case of discrete conditioning events

Description

Let A_1, \dots, A_p be p events forming a partition of a probability space and X_1, \dots, X_d be d random variables. Assume that we observe n i.i.d. replications of (X_1, \dots, X_d) , and that for each $i = 1, \dots, d$,

$$V_{i,j|A} = F_{X_j|A_k}(X_{i,j}|A_k),$$

we also know which of the A_k was realized. This function computes the pseudo-observations where k is such that the event A_k is realized for the i -th observation.

Usage

```
bCond.pobs(X, partition)
```

Arguments

`X` matrix of size $n \times d$ observations of conditioned variables.

`partition` matrix of size $n \times p$, where p is the number of conditioning events that are considered. `partition[i,k]` should be the indicator of whether the i -th observation belongs or not to the k -th conditioning event.

Value

a matrix of size $n \times d$ containing the conditional pseudo-observations $V_{i,j|A}$.

References

Derumigny, A., & Fermanian, J. D. (2017). About tests of the “simplifying” assumption for conditional copulas. *Dependence Modeling*, 5(1), 154-197. doi:10.1515/demo20170011

Derumigny, A., & Fermanian, J. D. (2022) Conditional empirical copula processes and generalized dependence measures *Electronic Journal of Statistics*, 16(2), 5692-5719. doi:10.1214/22EJS2075

See Also

[bCond.estParamCopula](#) for the estimation of a (conditional) parametric copula model in this framework.

[bCond.treeCKT](#) that provides a binary tree based on conditional Kendall’s tau and that can be used to derive relevant conditioning events.

Examples

```
n = 800
Z = stats::runif(n = n)
CKT = 0.2 * as.numeric(Z <= 0.3) +
  0.5 * as.numeric(Z > 0.3 & Z <= 0.5) +
  - 0.8 * as.numeric(Z > 0.5)
simCopula = VineCopula::BiCopSim(N = n,
  par = VineCopula::BiCopTau2Par(CKT, family = 1), family = 1)
X1 = simCopula[,1]
X2 = simCopula[,2]
partition = cbind(Z <= 0.3, Z > 0.3 & Z <= 0.5, Z > 0.5)
condPseudoObs = bCond.pobs(X = cbind(X1, X2),
  partition = partition)
```

bCond.simpA.CKT

Function for testing the simplifying assumption with data-driven box-type conditioning events

Description

This function takes in parameter the matrix of (observations) of the conditioned variables and either `matrixInd`, a matrix of indicator variables describing which events occur for which observations

Usage

```
bCond.simpA.CKT(
  XI,
  XJ = NULL,
  matrixInd = NULL,
  minCut = 0,
  minProb = 0.01,
  minSize = minProb * nrow(XI),
  nPoints_xJ = 10,
  type.quantile = 7,
  verbose = 2,
  methodTree = "doSplit",
  propTree = 0.5,
  methodPvalue = "bootNP",
  nBootstrap = 100
)
```

Arguments

XI	matrix of size $n \times p$ of observations of the conditioned variables.
XJ	matrix of size $n \times (d-p)$ containing observations of the conditioning vector.
matrixInd	a matrix of indexes of size $(n, N.\text{boxes})$ describing for each observation i to which box (= event) it belongs. If it is NULL, then a tree will be estimated to provide relevant boxes (by using bCond.treeCKT()) and then converting to a matrixInd by treeCKT2matrixInd() .
minCut	minimum difference in probabilities that is necessary to cut.
minProb	minimum probability of being in one of the node.
minSize	minimum number of observations in each node. This is an alternative to minProb and has priority over it.
nPoints_xJ	number of points in the grid that are considered when choosing the point for splitting the tree.
type.quantile	way of computing the quantiles, see stats::quantile() .
verbose	control the text output of the procedure. If verbose = 0, suppress all output. If verbose = 2, the progress of the computation is printed during the computation.
methodTree	method for constructing the tree <ul style="list-style-type: none"> doSplit some part of the data is used for constructing the tree and the other part for constructing the test statistic using the boxes defined by the estimated tree. The share of the data used for construction the tree is controlled by the parameter propTree. noSplit all of the data is used for both the tree and the test statistic on it. Note that p-values obtained by this method have an upward bias due to the lack of independence between these two steps. Only used if matrixInd is not provided.
propTree	share of observations used to build the tree (the rest of the observations are used for the computation of the p-value). Only used if matrixInd is not provided.


```

XI[-high_XJ1, ] = MASS::mvrnorm(n = n - length(high_XJ1), mu = c(8,8),
                                Sigma = rbind(c(1, -0.2), c(-0.2, 1)))

result = bCond.simpA.CKT(XI = XI, XJ = XJ, minSize = 10, verbose = 2,
                        methodTree = "doSplit", nBootstrap = 4)
print(result$p.value)
result2 = bCond.simpA.CKT(XI = XI, XJ = XJ, minSize = 10, verbose = 2,
                        methodTree = "noSplit", nBootstrap = 4)
print(result2$p.value)

```

bCond.simpA.param	<i>Test of the assumption that a conditional copulas does not vary through a list of discrete conditioning events</i>
-------------------	---

Description

Test of the assumption that a conditional copulas does not vary through a list of discrete conditioning events

Usage

```

bCond.simpA.param(
  X1,
  X2,
  partition,
  family,
  testStat = "T2c_tau",
  typeBoot = "boot.NP",
  nBootstrap = 100
)

```

Arguments

X1	vector of n observations of the first conditioned variable.
X2	vector of n observations of the second conditioned variable.
partition	matrix of size $n \times p$, where p is the number of conditioning events that are considered. <code>partition[i,j]</code> should be the indicator of whether the i -th observation belongs or not to the j -th conditioning event.
family	family of parametric copulas used
testStat	test statistic used. Possible choices are <ul style="list-style-type: none"> • <code>T2c_par</code> $\sum_{box} (\theta_0 - \theta(box))^2$ • <code>T2c_tau</code> Same as above, except that the copula family is now parametrized by its Kendall's tau instead of its natural parameter.
typeBoot	type of bootstrap used
nBootstrap	number of bootstrap replications

Value

a list containing

- `true_stat`: the value of the test statistic computed on the whole sample
- `vect_statB`: a vector of length `nBootstrap` containing the bootstrapped test statistics.
- `p_val`: the p-value of the test.

References

Derumigny, A., & Fermanian, J. D. (2017). About tests of the “simplifying” assumption for conditional copulas. *Dependence Modeling*, 5(1), 154-197. doi:10.1515/demo20170011

Derumigny, A., & Fermanian, J. D. (2022) Conditional empirical copula processes and generalized dependence measures *Electronic Journal of Statistics*, 16(2), 5692-5719. doi:10.1214/22EJS2075

See Also

[bCond.estParamCopula](#) for the estimation of a (conditional) parametric copula model in this framework.

[bCond.simpA.CKT](#) for a test of the simplifying assumption that all these conditional copulas are equal, based on the equality of conditional Kendall’s tau (i.e. without any parametric assumption).

Tests of the simplifying assumption for conditional copulas with a continuous conditioning variable:

- [simpA.NP](#) in a nonparametric setting
- [simpA.param](#) in a (semi)parametric setting, where the conditional copula belongs to a parametric family, but the conditional margins are estimated arbitrarily through kernel smoothing
- [simpA.kendallReg](#): test based on the constancy of conditional Kendall’s tau

Examples

```
n = 50 # to get fast computation time. More realistic is n = 800.

Z = stats::runif(n = n)
CKT = 0.2 * as.numeric(Z <= 0.3) +
  0.5 * as.numeric(Z > 0.3 & Z <= 0.5) +
  0.3 * as.numeric(Z > 0.5)
family = 3
simCopula = VineCopula::BiCopSim(N = n,
  par = VineCopula::BiCopTau2Par(CKT, family = family), family = family)
X1 = simCopula[,1]
X2 = simCopula[,2]
partition = cbind(Z <= 0.3, Z > 0.3 & Z <= 0.5, Z > 0.5)

result = bCond.simpA.param(X1 = X1, X2 = X2, testStat = "T2c_tau",
  partition = partition, family = family, typeBoot = "boot.paramInd")
print(result$p_val)

n = 800
Z = stats::runif(n = n)
CKT = 0.1
```

```

family = 3
simCopula = VineCopula::BiCopSim(N = n,
  par = VineCopula::BiCopTau2Par(CKT, family = family), family = family)
X1 = simCopula[,1]
X2 = simCopula[,2]
partition = cbind(Z <= 0.3, Z > 0.3 & Z <= 0.5, Z > 0.5)

result = bCond.simpA.param(X1 = X1, X2 = X2,
  partition = partition, family = family, typeBoot = "boot.NP")
print(result$p_val)

```

bCond.treeCKT

Construct a binary tree for the modeling the conditional Kendall's tau

Description

This function takes in parameter two matrices of observations: the first one contains the observations of XI (the conditioned variables) and the second one contains the observations of XJ (the conditioning variables). The goal of this procedure is to find which of the variables in XJ have important influence on the dependence between the components of XI, (measured by the Kendall's tau).

Usage

```

bCond.treeCKT(
  XI,
  XJ,
  minCut = 0,
  minProb = 0.01,
  minSize = minProb * nrow(XI),
  nPoints_xJ = 10,
  type.quantile = 7,
  verbose = 2
)

```

Arguments

XI	matrix of size $n \times p$ of observations of the conditioned variables.
XJ	matrix of size $n \times (d-p)$ containing observations of the conditioning vector.
minCut	minimum difference in probabilities that is necessary to cut.
minProb	minimum probability of being in one of the node.
minSize	minimum number of observations in each node. This is an alternative to minProb and has priority over it.
nPoints_xJ	number of points in the grid that are considered when choosing the point for splitting the tree.

`type.quantile` way of computing the quantiles, see `stats::quantile()`.
`verbose` control the text output of the procedure. If `verbose = 0`, suppress all output. If `verbose = 2`, the progress of the computation is printed during the computation.

Details

The object return by this function is a binary tree. Each leaf of this tree correspond to one event (or, equivalently, one subset of $R^{dim(X^J)}$), and the conditional Kendall's tau conditionally to it.

Value

the estimated tree using the data 'XI, XJ'.

References

Derumigny, A., Fermanian, J. D., & Min, A. (2022). Testing for equality between conditional copulas given discretized conditioning events. Canadian Journal of Statistics. doi:10.1002/cjs.11742

See Also

[bCond.simpA.CKT](#) for a test of the simplifying assumption that all these conditional Kendall's tau are equal.

[treeCKT2matrixInd](#) for converting this tree to a matrix of indicators of each event. [matrixInd2matrixCKT](#) for getting the matrix of estimated conditional Kendall's taus for each event.

[CKT.estimate](#) for the estimation of pointwise conditional Kendall's tau, i.e. assuming a continuous conditioning variable Z .

Examples

```
set.seed(1)
n = 400
XJ = MASS::mvrnorm(n = n, mu = c(3,3), Sigma = rbind(c(1, 0.2), c(0.2, 1)))
XI = matrix(nrow = n, ncol = 2)
high_XJ1 = which(XJ[,1] > 4)
XI[high_XJ1, ] = MASS::mvrnorm(n = length(high_XJ1), mu = c(10,10),
                               Sigma = rbind(c(1, 0.8), c(0.8, 1)))
XI[-high_XJ1, ] = MASS::mvrnorm(n = n - length(high_XJ1), mu = c(8,8),
                               Sigma = rbind(c(1, -0.2), c(-0.2, 1)))

result = bCond.treeCKT(XI = XI, XJ = XJ, minSize = 50, verbose = 2)
# Plotting the corresponding tree using the "DiagrammeR" package
if (requireNamespace("DiagrammeR", quietly = TRUE)){
  plot(result)
}

# Number of observations in the first two children
print(length(data.tree::GetAttribute(result$children[[1]], "condObs")))
print(length(data.tree::GetAttribute(result$children[[2]], "condObs")))
```

CKT.estimate	<i>Estimation of conditional Kendall's tau between two variables X1 and X2 given Z = z</i>
--------------	--

Description

Let X_1 and X_2 be two random variables. The goal of this function is to estimate the conditional Kendall's tau (a dependence measure) between X_1 and X_2 given $Z = z$ for a conditioning variable Z . Conditional Kendall's tau between X_1 and X_2 given $Z = z$ is defined as:

$$P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) > 0 | Z_1 = Z_2 = z) \\ - P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) < 0 | Z_1 = Z_2 = z),$$

where $(X_{1,1}, X_{1,2}, Z_1)$ and $(X_{2,1}, X_{2,2}, Z_2)$ are two independent and identically distributed copies of (X_1, X_2, Z) . In other words, conditional Kendall's tau is the difference between the probabilities of observing concordant and discordant pairs from the conditional law of

$$(X_1, X_2) | Z = z.$$

This function can use different estimators for conditional Kendall's tau, see the description of the parameter `methodEstimation` for a complete list of possibilities.

Usage

```
CKT.estimate(
  X1 = NULL, X2 = NULL, Z = NULL,
  newZ = Z, methodEstimation, h,
  listPhi = if(methodEstimation == "kendallReg")
    {list( function(x){return(x)} ,
          function(x){return(x^2)} ,
          function(x){return(x^3)} )
    } else {list(identity)} ,
  ... ,
  observedX1 = NULL, observedX2 = NULL, observedZ = NULL )
```

Arguments

- | | |
|------|--|
| X1 | a vector of n observations of the first variable |
| X2 | a vector of n observations of the second variable |
| Z | a vector of n observations of the conditioning variable, or a matrix with n rows of observations of the conditioning vector (if Z is multivariate). |
| newZ | the new values for the conditioning variable Z at which the conditional Kendall's tau should be estimated. <ul style="list-style-type: none"> • If <code>observedZ</code> is a vector, then <code>newZ</code> must be a vector as well. • If <code>observedZ</code> is a matrix, then <code>newZ</code> must be a matrix as well, with the same number of columns (= the dimension of Z). |

methodEstimation	<p>method for estimating the conditional Kendall's tau. Possible estimation methods are:</p> <ul style="list-style-type: none"> • "kernel": kernel smoothing, as described in (Derumigny, & Fermanian (2019a)) • "kendallReg": regression-type model, as described in (Derumigny, & Fermanian (2020)) • "tree", "randomForest", "logit", and "neuralNetwork": use the relationship between conditional Kendall's tau and classification problems to use the respective classification algorithms for the estimation of conditional Kendall's tau, as described in (Derumigny, & Fermanian (2019b))
h	the bandwidth
listPhi	the list of transformations to be applied to the conditioning variable Z (in case of regression-type models).
...	other parameters passed to the estimating functions CKT.fit.tree , CKT.fit.randomForest , CKT.fit.GLM , CKT.fit.nNets , CKT.predict.kNN , CKT.kernel and CKT.kendallReg.fit .
observedX1, observedX2, observedZ	old parameter names for X_1 , X_2 , Z . Support for this will be removed at a later version.

Value

the vector of estimated conditional Kendall's tau at each of the observations of newZ.

References

- Derumigny, A., & Fermanian, J. D. (2019a). A classification point-of-view about conditional Kendall's tau. *Computational Statistics & Data Analysis*, 135, 70-94. doi:[10.1016/j.csda.2019.01.013](#)
- Derumigny, A., & Fermanian, J. D. (2019b). On kernel-based estimation of conditional Kendall's tau: finite-distance bounds and asymptotic behavior. *Dependence Modeling*, 7(1), 292-321. doi:[10.1515/demo20190016](#)
- Derumigny, A., & Fermanian, J. D. (2020). On Kendall's regression. *Journal of Multivariate Analysis*, 178, 104610. doi:[10.1016/j.jmva.2020.104610](#)

See Also

the specialized functions for estimating conditional Kendall's tau for each method: [CKT.fit.tree](#), [CKT.fit.randomForest](#), [CKT.fit.GLM](#), [CKT.fit.nNets](#), [CKT.predict.kNN](#), [CKT.fit.randomForest](#), [CKT.kernel](#) and [CKT.kendallReg.fit](#).

See also the nonparametric estimator of conditional copula models [estimateNPCondCopula](#), and the parametric estimators of conditional copula models [estimateParCondCopula](#).

In the case where Z is discrete or in the case of discrete conditioning events, see [bCond.treeCKT](#).

Examples

```
# We simulate from a conditional copula
set.seed(1)
N = 300
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = -0.9 + 1.8 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

newZ = seq(2,8,by = 0.1)
h = 0.1
estimatedCKT_tree <- CKT.estimate(
  X1 = X1, X2 = X2, Z = Z,
  newZ = newZ,
  methodEstimation = "tree", h = h)

estimatedCKT_rf <- CKT.estimate(
  X1 = X1, X2 = X2, Z = Z,
  newZ = newZ,
  methodEstimation = "randomForest", h = h)

estimatedCKT_GLM <- CKT.estimate(
  X1 = X1, X2 = X2, Z = Z,
  newZ = newZ,
  methodEstimation = "logit", h = h,
  listPhi = list(function(x){return(x)}, function(x){return(x^2)},
    function(x){return(x^3)}) )

estimatedCKT_kNN <- CKT.estimate(
  X1 = X1, X2 = X2, Z = Z,
  newZ = newZ,
  methodEstimation = "nearestNeighbors", h = h,
  number_nn = c(50,80, 100, 120,200),
  partition = 4
)

estimatedCKT_nNet <- CKT.estimate(
  X1 = X1, X2 = X2, Z = Z,
  newZ = newZ,
  methodEstimation = "neuralNetwork", h = h,
)

estimatedCKT_kernel <- CKT.estimate(
  X1 = X1, X2 = X2, Z = Z,
  newZ = newZ,
  methodEstimation = "kernel", h = h,
)

estimatedCKT_kendallReg <- CKT.estimate(
  X1 = X1, X2 = X2, Z = Z,
```

```

newZ = newZ,
methodEstimation = "kendallReg", h = h)

# Comparison between true Kendall's tau (in black)
# and estimated Kendall's tau (in other colors)
trueConditionalTau = -0.9 + 1.8 * pnorm(newZ, mean = 5, sd = 2)
plot(newZ, trueConditionalTau, col="black",
     type = "l", ylim = c(-1, 1))
lines(newZ, estimatedCKT_tree, col = "red")
lines(newZ, estimatedCKT_rf, col = "blue")
lines(newZ, estimatedCKT_GLM, col = "green")
lines(newZ, estimatedCKT_kNN, col = "purple")
lines(newZ, estimatedCKT_nNet, col = "coral")
lines(newZ, estimatedCKT_kernel, col = "skyblue")
lines(newZ, estimatedCKT_kendallReg, col = "darkgreen")

```

CKT.fit.GLM

Estimation of conditional Kendall's taus by penalized GLM

Description

The function `CKT.fit.GLM` fits a regression model for the conditional Kendall's tau $\tau_{1,2|Z}$ between two variables X_1 and X_2 conditionally to some predictors Z . More precisely, this function fits the model

$$\tau_{1,2|Z} = 2 * \Lambda(\beta_0 + \beta_1 \phi_1(Z) + \dots + \beta_p \phi_p(Z))$$

for a link function Λ , and p real-valued functions ϕ_1, \dots, ϕ_p . The function `CKT.predict.GLM` predicts the values of conditional Kendall's tau for some values of the conditioning variable Z .

Usage

```

CKT.fit.GLM(
  datasetPairs,
  designMatrix = datasetPairs[, 2:(ncol(datasetPairs) - 3), drop = FALSE],
  link = "logit",
  ...
)

CKT.predict.GLM(fit, newZ)

```

Arguments

<code>datasetPairs</code>	the matrix of pairs and corresponding values of the kernel as provided by datasetPairs .
<code>designMatrix</code>	the matrix of predictor to be used for the fitting of the model. It should have the same number of rows as the <code>datasetPairs</code> .
<code>link</code>	link function, can be one of <code>logit</code> , <code>probit</code> , <code>cloglog</code> , <code>cauchit</code> .
<code>...</code>	other parameters passed to <code>ordinalNet::ordinalNet()</code> .

<code>fit</code>	result of a call to <code>CKT.fit.GLM</code>
<code>newZ</code>	new matrix of observations of the conditioning vector Z , with the same number of variables and same names as the <code>designMatrix</code> that was used to fit the GLM.

Value

`CKT.fit.GLM` returns the fitted GLM, an object with S3 class `ordinalNet`.

`CKT.predict.GLM` returns a vector of (predicted) conditional Kendall's taus of the same size as the number of rows of the matrix `newZ`.

References

Derumigny, A., & Fermanian, J. D. (2019). A classification point-of-view about conditional Kendall's tau. *Computational Statistics & Data Analysis*, 135, 70-94. (Algorithm 2) doi:[10.1016/j.csda.2019.01.013](https://doi.org/10.1016/j.csda.2019.01.013)

See Also

See also other estimators of conditional Kendall's tau: [CKT.fit.tree](#), [CKT.fit.randomForest](#), [CKT.fit.nNets](#), [CKT.predict.kNN](#), [CKT.kernel](#), [CKT.kendallReg.fit](#), and the more general wrapper [CKT.estimate](#).

Examples

```
# We simulate from a conditional copula
set.seed(1)
N = 400
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = 2*plogis(-1 + 0.8*Z - 0.1*Z^2) - 1
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

datasetP = datasetPairs(X1 = X1, X2 = X2, Z = Z, h = 0.07, cut = 0.9)
designMatrix = cbind(datasetP[,2], datasetP[,2]^2)
fitCKT_GLM <- CKT.fit.GLM(
  datasetPairs = datasetP, designMatrix = designMatrix,
  maxiterOut = 10, maxiterIn = 5)
print(coef(fitCKT_GLM))
# These are rather close to the true coefficients -1, 0.8, -0.1
# used to generate the data above.

newZ = seq(2,10,by = 0.1)
estimatedCKT_GLM = CKT.predict.GLM(
  fit = fitCKT_GLM, newZ = cbind(newZ, newZ^2))

# Comparison between true Kendall's tau (in red)
# and estimated Kendall's tau (in black)
trueConditionalTau = 2*plogis(-1 + 0.8*newZ - 0.1*newZ^2) - 1
plot(newZ, trueConditionalTau, col="red",
  type = "l", ylim = c(-1, 1))
```



```
lines(newZ, estimatedCKT_GLM)
```

CKT.fit.nNets	<i>Estimation of conditional Kendall's taus by model averaging of neural networks</i>
---------------	---

Description

Let X_1 and X_2 be two random variables. The goal of this function is to estimate the conditional Kendall's tau (a dependence measure) between X_1 and X_2 given $Z = z$ for a conditioning variable Z . Conditional Kendall's tau between X_1 and X_2 given $Z = z$ is defined as:

$$P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) > 0 | Z_1 = Z_2 = z) \\ - P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) < 0 | Z_1 = Z_2 = z),$$

where $(X_{1,1}, X_{1,2}, Z_1)$ and $(X_{2,1}, X_{2,2}, Z_2)$ are two independent and identically distributed copies of (X_1, X_2, Z) . In other words, conditional Kendall's tau is the difference between the probabilities of observing concordant and discordant pairs from the conditional law of

$$(X_1, X_2) | Z = z.$$

This function estimates conditional Kendall's tau using **neural networks**. This is possible by the relationship between estimation of conditional Kendall's tau and classification problems (see Derumigny and Fermanian (2019)): estimation of conditional Kendall's tau is equivalent to the prediction of concordance in the space of pairs of observations.

Usage

```
CKT.fit.nNets(
  datasetPairs,
  designMatrix = datasetPairs[, 2:(ncol(datasetPairs) - 3), drop = FALSE],
  vecSize = rep(3, times = 10),
  nObs_per_NN = 0.9 * nrow(designMatrix),
  verbose = 1
)
```

Arguments

datasetPairs	the matrix of pairs and corresponding values of the kernel as provided by datasetPairs .
designMatrix	the matrix of predictor to be used for the fitting of the tree
vecSize	vector with the number of neurons for each network
nObs_per_NN	number of observations used for each neural network.
verbose	a number indicated what to print <ul style="list-style-type: none"> • 0: nothing printed at all. • 1: a message is printed at the convergence of each neural network. • 2: details are printed for each optimization of each network.

Value

CKT.fit.nNets returns a list of the fitted neural networks

References

Derumigny, A., & Fermanian, J. D. (2019). A classification point-of-view about conditional Kendall's tau. Computational Statistics & Data Analysis, 135, 70-94. (Algorithm 7) doi:[10.1016/j.csda.2019.01.013](https://doi.org/10.1016/j.csda.2019.01.013)

See Also

See also other estimators of conditional Kendall's tau: [CKT.fit.tree](#), [CKT.fit.randomForest](#), [CKT.fit.GLM](#), [CKT.predict.kNN](#), [CKT.kernel](#), [CKT.kendallReg.fit](#), and the more general wrapper [CKT.estimate](#).

Examples

```
# We simulate from a conditional copula
set.seed(1)
N = 800
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = -0.9 + 1.8 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

newZ = seq(2,10,by = 0.1)
datasetP = datasetPairs(X1 = X1, X2 = X2, Z = Z, h = 0.07, cut = 0.9)

fitCKT_nets <- CKT.fit.nNets(datasetPairs = datasetP)
estimatedCKT_nNets <- CKT.predict.nNets(
  fit = fitCKT_nets, newZ = matrix(newZ, ncol = 1))

# Comparison between true Kendall's tau (in black)
# and estimated Kendall's tau (in red)
trueConditionalTau = -0.9 + 1.8 * pnorm(newZ, mean = 5, sd = 2)
plot(newZ, trueConditionalTau, col="black",
  type = "l", ylim = c(-1, 1))
lines(newZ, estimatedCKT_nNets, col = "red")
```

CKT.fit.randomForest	<i>Fit a Random Forest that can be used for the estimation of conditional Kendall's tau.</i>
----------------------	--

Description

Let X_1 and X_2 be two random variables. The goal of this function is to estimate the conditional Kendall's tau (a dependence measure) between X_1 and X_2 given $Z = z$ for a conditioning variable Z . Conditional Kendall's tau between X_1 and X_2 given $Z = z$ is defined as:

$$P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) > 0 | Z_1 = Z_2 = z) \\ - P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) < 0 | Z_1 = Z_2 = z),$$

where $(X_{1,1}, X_{1,2}, Z_1)$ and $(X_{2,1}, X_{2,2}, Z_2)$ are two independent and identically distributed copies of (X_1, X_2, Z) . In other words, conditional Kendall's tau is the difference between the probabilities of observing concordant and discordant pairs from the conditional law of

$$(X_1, X_2) | Z = z.$$

These functions estimate and predict conditional Kendall's tau using a **random forest**. This is possible by the relationship between estimation of conditional Kendall's tau and classification problems (see Derumigny and Fermanian (2019)): estimation of conditional Kendall's tau is equivalent to the prediction of concordance in the space of pairs of observations.

Usage

```
CKT.fit.randomForest(
  datasetPairs,
  designMatrix = data.frame(x = datasetPairs[, 2:(ncol(datasetPairs) - 3)]),
  n,
  nTree = 10,
  mindev = 0.008,
  mincut = 0,
  nObs_per_Tree = ceiling(0.8 * n),
  nVar_per_Tree = ceiling(0.8 * (ncol(datasetPairs) - 4)),
  verbose = FALSE,
  nMaxDepthAllowed = 10
)

CKT.predict.randomForest(fit, newZ)
```

Arguments

datasetPairs	the matrix of pairs and corresponding values of the kernel as provided by datasetPairs .
designMatrix	the matrix of predictor to be used for the fitting of the tree
n	the original sample size of the dataset
nTree	number of trees of the Random Forest.
mindev	a factor giving the minimum deviation for a node to be splitted. See <code>tree::tree.control()</code> for more details.
mincut	the minimum number of observations (of pairs) in a node See <code>tree::tree.control()</code> for more details.
nObs_per_Tree	number of observations kept in each tree.

nVar_per_Tree number of variables kept in each tree.

verbose if TRUE, a message is printed after fitting each tree.

nMaxDepthAllowed the maximum number of errors of type "the tree cannot be fitted" or "is too deep" before stopping the procedure.

fit result of a call to CKT.fit.randomForest.

newZ new matrix of observations, with the same number of variables. and same names as the designMatrix that was used to fit the Random Forest.

Value

a list with two components

- `list_tree` a list of size `nTree` composed of all the fitted trees.
- `list_variables` a list of size `nTree` composed of the (predictor) variables for each tree.

`CKT.predict.randomForest` returns a vector of (predicted) conditional Kendall's taus of the same size as the number of rows of the `newZ`.

References

Derumigny, A., & Fermanian, J. D. (2019). A classification point-of-view about conditional Kendall's tau. *Computational Statistics & Data Analysis*, 135, 70-94. (Algorithm 4) [doi:10.1016/j.csda.2019.01.013](https://doi.org/10.1016/j.csda.2019.01.013)

Examples

```
# We simulate from a conditional copula
set.seed(1)
N = 800
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = -0.9 + 1.8 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N , family = 1,
  par = VineCopula::BiCopTau2Par(1 , conditionalTau ))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

datasetP = datasetPairs(X1 = X1, X2 = X2, Z = Z, h = 0.07, cut = 0.9)
est_RF = CKT.fit.randomForest(datasetPairs = datasetP, n = N,
  mindev = 0.008)

newZ = seq(1,10,by = 0.1)
prediction = CKT.predict.randomForest(fit = est_RF,
  newZ = data.frame(x=newZ))
# Comparison between true Kendall's tau (in red)
# and estimated Kendall's tau (in black)
plot(newZ, prediction, type = "l", ylim = c(-1,1))
lines(newZ, -0.9 + 1.8 * pnorm(newZ, mean = 5, sd = 2), col="red")
```

Description

Let X_1 and X_2 be two random variables. The goal of this function is to estimate the conditional Kendall's tau (a dependence measure) between X_1 and X_2 given $Z = z$ for a conditioning variable Z . Conditional Kendall's tau between X_1 and X_2 given $Z = z$ is defined as:

$$P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) > 0 | Z_1 = Z_2 = z) \\ - P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) < 0 | Z_1 = Z_2 = z),$$

where $(X_{1,1}, X_{1,2}, Z_1)$ and $(X_{2,1}, X_{2,2}, Z_2)$ are two independent and identically distributed copies of (X_1, X_2, Z) . In other words, conditional Kendall's tau is the difference between the probabilities of observing concordant and discordant pairs from the conditional law of

$$(X_1, X_2) | Z = z.$$

These functions estimate and predict conditional Kendall's tau using a **classification tree**. This is possible by the relationship between estimation of conditional Kendall's tau and classification problems (see Derumigny and Fermanian (2019)): estimation of conditional Kendall's tau is equivalent to the prediction of concordance in the space of pairs of observations.

Usage

```
CKT.fit.tree(datasetPairs, mindev = 0.008, mincut = 0)
```

```
CKT.predict.tree(fit, newZ)
```

Arguments

datasetPairs	the matrix of pairs and corresponding values of the kernel as provided by datasetPairs .
mindev	a factor giving the minimum deviation for a node to be splitted. See <code>tree::tree.control()</code> for more details.
mincut	the minimum number of observations (of pairs) in a node See <code>tree::tree.control()</code> for more details.
fit	result of a call to <code>CKT.fit.tree</code>
newZ	new matrix of observations, with the same number of variables. and same names as the <code>designMatrix</code> that was used to fit the tree.

Value

`CKT.fit.tree` returns the fitted tree.

`CKT.predict.tree` returns a vector of (predicted) conditional Kendall's taus of the same size as the number of rows of `newZ`.

References

Derumigny, A., & Fermanian, J. D. (2019). A classification point-of-view about conditional Kendall's tau. Computational Statistics & Data Analysis, 135, 70-94. (Section 3.2) [doi:10.1016/j.csda.2019.01.013](https://doi.org/10.1016/j.csda.2019.01.013)

See Also

See also other estimators of conditional Kendall's tau: [CKT.fit.nNets](#), [CKT.fit.randomForest](#), [CKT.fit.GLM](#), [CKT.predict.kNN](#), [CKT.kernel](#), [CKT.kendallReg.fit](#), and the more general wrapper [CKT.estimate](#).

Examples

```
# We simulate from a conditional copula
set.seed(1)
N = 800
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = -0.9 + 1.8 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N , family = 1,
  par = VineCopula::BiCopTau2Par(1 , conditionalTau ))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

datasetP = datasetPairs(X1 = X1, X2 = X2, Z = Z, h = 0.07, cut = 0.9)
est_Tree = CKT.fit.tree(datasetPairs = datasetP, mindev = 0.008)
print(est_Tree)

newZ = seq(1,10,by = 0.1)
prediction = CKT.predict.tree(fit = est_Tree, newZ = data.frame(x=newZ))
# Comparison between true Kendall's tau (in red)
# and estimated Kendall's tau (in black)
plot(newZ, prediction, type = "l", ylim = c(-1,1))
lines(newZ, -0.9 + 1.8 * pnorm(newZ, mean = 5, sd = 2), col="red")
```

CKT.hCV.l1out

Choose the bandwidth for kernel estimation of conditional Kendall's tau using cross-validation

Description

Let X_1 and X_2 be two random variables. The goal here is to estimate the conditional Kendall's tau (a dependence measure) between X_1 and X_2 given $Z = z$ for a conditioning variable Z . Conditional Kendall's tau between X_1 and X_2 given $Z = z$ is defined as:

$$P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) > 0 | Z_1 = Z_2 = z) \\ - P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) < 0 | Z_1 = Z_2 = z),$$

where $(X_{1,1}, X_{1,2}, Z_1)$ and $(X_{2,1}, X_{2,2}, Z_2)$ are two independent and identically distributed copies of (X_1, X_2, Z) . For this, a kernel-based estimator is used, as described in (Derumigny & Fermanian

(2019)). These functions aims at finding the best bandwidth h among a given `range_h` by cross-validation. They use either:

- **leave-one-out** cross-validation: function `CKT.hCV.l1out`
- or **K-folds** cross-validation: function `CKT.hCV.Kfolds`

Usage

```
CKT.hCV.l1out(
  X1 = NULL,
  X2 = NULL,
  Z = NULL,
  range_h,
  matrixSignsPairs = NULL,
  nPairs = NULL,
  typeEstCKT = "wdm",
  kernel.name = "Epa",
  progressBar = TRUE,
  verbose = FALSE,
  observedX1 = NULL,
  observedX2 = NULL,
  observedZ = NULL
)
```

```
CKT.hCV.Kfolds(
  X1,
  X2,
  Z,
  ZToEstimate,
  range_h,
  matrixSignsPairs = NULL,
  typeEstCKT = "wdm",
  kernel.name = "Epa",
  Kfolds = 5,
  progressBar = TRUE,
  verbose = FALSE,
  observedX1 = NULL,
  observedX2 = NULL,
  observedZ = NULL
)
```

Arguments

<code>X1</code>	a vector of n observations of the first variable
<code>X2</code>	a vector of n observations of the second variable
<code>Z</code>	vector of observed values of Z . If Z is multivariate, then this is a matrix whose rows correspond to the observations of Z
<code>range_h</code>	vector containing possible values for the bandwidth.

<code>matrixSignsPairs</code>	square matrix of signs of all pairs, produced by <code>computeMatrixSignPairs</code> (<code>observedX1</code> , <code>observedX2</code>). Only needed if <code>typeEstCKT</code> is not the default 'wdm'.
<code>nPairs</code>	number of pairs used in the cross-validation criteria. Use <code>nPairs = "all"</code> to choose all pairs. The default is <code>nPairs = 10 * n</code> , where <code>n</code> is the sample size.
<code>typeEstCKT</code>	type of estimation of the conditional Kendall's tau.
<code>kernel.name</code>	name of the kernel used for smoothing. Possible choices are "Gaussian" (Gaussian kernel) and "Epa" (Epanechnikov kernel).
<code>progressBar</code>	if TRUE, a progressbar for each <code>h</code> is displayed to show the progress of the computation.
<code>verbose</code>	if TRUE, print the score of each <code>h</code> during the procedure.
<code>observedX1, observedX2, observedZ</code>	old parameter names for <code>X1</code> , <code>X2</code> , <code>Z</code> . Support for this will be removed at a later version.
<code>ZToEstimate</code>	vector of fixed conditioning values at which the difference between the two conditional Kendall's tau should be computed. Can also be a matrix whose lines are the conditioning vectors at which the difference between the two conditional Kendall's tau should be computed.
<code>Kfolds</code>	number of subsamples used.

Value

Both functions return a list with two components:

- `hCV`: the chosen bandwidth
- `scores`: vector of the same length as `range_h` giving the value of the CV criteria for each of the `h` tested. Lower score indicates a better fit.

References

Derumigny, A., & Fermanian, J. D. (2019). On kernel-based estimation of conditional Kendall's tau: finite-distance bounds and asymptotic behavior. *Dependence Modeling*, 7(1), 292-321. Page 296, Equation (4). doi:[10.1515/demo20190016](https://doi.org/10.1515/demo20190016)

See Also

[CKT.kernel](#) for the corresponding estimator of conditional Kendall's tau by kernel smoothing.

Examples

```
# We simulate from a conditional copula
set.seed(1)
N = 200
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = -0.9 + 1.8 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1])
```



```

X2 = qnorm(simCopula[,2])

newZ = seq(2,10,by = 0.1)
range_h = 3:10

resultCV <- CKT.hCV.l1out(X1 = X1, X2 = X2, Z = Z,
                        range_h = range_h, nPairs = 100)

resultCV <- CKT.hCV.Kfolds(X1 = X1, X2 = X2, Z = Z,
                        range_h = range_h, ZToEstimate = newZ)

plot(range_h, resultCV$scores, type = "b")

```

CKT.kendallReg.fit	<i>Fit Kendall's regression, a GLM-type model for conditional Kendall's tau</i>
--------------------	---

Description

The function `CKT.kendallReg.fit` fits a regression-type model for the conditional Kendall's tau between two variables X_1 and X_2 conditionally to some predictors Z . More precisely, it fits the model

$$\Lambda(\tau_{X_1, X_2|Z=z}) = \sum_{j=1}^{p'} \beta_j \psi_j(z),$$

where $\tau_{X_1, X_2|Z=z}$ is the conditional Kendall's tau between X_1 and X_2 conditionally to $Z = z$, Λ is a function from $] - 1, 1]$ to R , $(\beta_1, \dots, \beta_{p'})$ are unknown coefficients to be estimated and $\psi_1, \dots, \psi_{p'}$ are a dictionary of functions. To estimate *beta*, we used the penalized estimator which is defined as the minimizer of the following criteria

$$\frac{1}{2n'} \sum_{i=1}^{n'} [\Lambda(\hat{\tau}_{X_1, X_2|Z=z_i}) - \sum_{j=1}^{p'} \beta_j \psi_j(z_i)]^2 + \lambda * |\beta|_1,$$

where the z_i are a second sample (here denoted by `ZToEstimate`).

The function `CKT.kendallReg.predict` predicts the conditional Kendall's tau between two variables X_1 and X_2 given $Z = z$ for some new values of z .

Usage

```

CKT.kendallReg.fit(
  X1 = NULL,
  X2 = NULL,
  Z = NULL,
  ZToEstimate,
  designMatrixZ = cbind(ZToEstimate, ZToEstimate^2, ZToEstimate^3),
  newZ = designMatrixZ,

```

```

    h_kernel,
    Lambda = identity,
    Lambda_inv = identity,
    lambda = NULL,
    Kfolds_lambda = 10,
    l_norm = 1,
    h_lambda = h_kernel,
    ...,
    observedX1 = NULL,
    observedX2 = NULL,
    observedZ = NULL
)

CKT.kendallReg.predict(fit, newZ, lambda = NULL, Lambda_inv = identity)

```

Arguments

<code>X1</code>	a vector of n observations of the first variable X_1 .
<code>X2</code>	a vector of n observations of the second variable X_2 .
<code>Z</code>	a vector of n observations of the conditioning variable, or a matrix with n rows of observations of the conditioning vector (if Z is multivariate).
<code>ZToEstimate</code>	the intermediary dataset of observations of Z at which the conditional Kendall's tau should be estimated.
<code>designMatrixZ</code>	the transformation of the <code>ZToEstimate</code> that will be used as predictors. By default, no transformation is applied.
<code>newZ</code>	the new observations of the conditioning variable.
<code>h_kernel</code>	bandwidth used for the first step of kernel smoothing.
<code>Lambda</code>	the function to be applied on conditional Kendall's tau. By default, the identity function is used.
<code>Lambda_inv</code>	the functional inverse of <code>Lambda</code> . By default, the identity function is used.
<code>lambda</code>	the regularization parameter. If <code>NULL</code> , then it is chosen by K-fold cross validation. Internally, cross-validation is performed by the function CKT.KendallReg.LambdaCV .
<code>Kfolds_lambda</code>	the number of folds used in the cross-validation procedure to choose <code>lambda</code> .
<code>l_norm</code>	type of norm used for selection of the optimal <code>lambda</code> by cross-validation. <code>l_norm=1</code> corresponds to the sum of absolute values of differences between predicted and estimated conditional Kendall's tau while <code>l_norm=2</code> corresponds to the sum of squares of differences.
<code>h_lambda</code>	the smoothing bandwidth used in the cross-validation procedure to choose <code>lambda</code> .
<code>...</code>	other arguments to be passed to CKT.kernel for the first step (kernel-based) estimator of conditional Kendall's tau.
<code>observedX1, observedX2, observedZ</code>	old parameter names for X_1 , X_2 , Z . Support for this will be removed at a later version.
<code>fit</code>	the fitted model, obtained by a call to <code>CKT.kendallReg.fit</code> .

Value

The function `CKT.kendallReg.fit` returns a list with the following components:

- `estimatedCKT`: the estimated CKT at the new data points `newZ`.
- `fit`: the fitted model, of S3 class `glmnet` (see `glmnet::glmnet` for more details).
- `lambda`: the value of the penalized parameter used. (i.e. either the one supplied by the user or the one determined by cross-validation)

`CKT.kendallReg.predict` returns the predicted values of conditional Kendall's tau.

References

Derumigny, A., & Fermanian, J. D. (2020). On Kendall's regression. *Journal of Multivariate Analysis*, 178, 104610. doi:10.1016/j.jmva.2020.104610

See Also

See also other estimators of conditional Kendall's tau: `CKT.fit.tree`, `CKT.fit.randomForest`, `CKT.fit.nNets`, `CKT.predict.kNN`, `CKT.kernel`, `CKT.fit.GLM`, and the more general wrapper `CKT.estimate`.

See also the test of the simplifying assumption that a conditional copula does not depend on the value of the conditioning variable using the nullity of Kendall's regression coefficients: `simpA.kendallReg`.

Examples

```
# We simulate from a conditional copula
set.seed(1)
N = 400
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = -0.9 + 1.8 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

newZ = seq(2, 10, by = 0.1)
estimatedCKT_kendallReg <- CKT.kendallReg.fit(
  X1 = X1, X2 = X2, Z = Z,
  ZToEstimate = newZ, h_kernel = 0.07)

coef(estimatedCKT_kendallReg$fit,
  s = estimatedCKT_kendallReg$lambda)

# Comparison between true Kendall's tau (in black)
# and estimated Kendall's tau (in red)
trueConditionalTau = -0.9 + 1.8 * pnorm(newZ, mean = 5, sd = 2)
plot(newZ, trueConditionalTau, col="black",
  type = "l", ylim = c(-1, 1))
lines(newZ, estimatedCKT_kendallReg$estimatedCKT, col = "red")
```

 CKT.KendallReg.LambdaCV

Kendall's regression: choice of the penalization parameter by K-folds cross-validation

Description

In this model, three variables X_1 , X_2 and Z are observed. We try to model the conditional Kendall's tau between X_1 and X_2 conditionally to $Z = z$, as follows:

$$\Lambda(\tau_{X_1, X_2|Z=z}) = \sum_{i=1}^{p'} \beta_i \psi_i(z),$$

where $\tau_{X_1, X_2|Z=z}$ is the conditional Kendall's tau between X_1 and X_2 conditionally to $Z = z$, Λ is a function from $] - 1, 1[$ to R , $(\beta_1, \dots, \beta_{p'})$ are unknown coefficients to be estimated and $\psi_1, \dots, \psi_{p'}$ are a dictionary of functions. To estimate β , we used the penalized estimator which is defined as the minimizer of the following criteria

$$\frac{1}{2n'} \sum_{i=1}^{n'} [\Lambda(\hat{\tau}_{X_1, X_2|Z=z}) - \sum_{j=1}^{p'} \beta_j \psi_j(z)]^2 + \lambda * |\beta|_1.$$

This function chooses the penalization parameter *lambda* by cross-validation.

Usage

```
CKT.KendallReg.LambdaCV(
  X1 = NULL,
  X2 = NULL,
  Z = NULL,
  ZToEstimate,
  designMatrixZ = cbind(ZToEstimate, ZToEstimate^2, ZToEstimate^3),
  typeEstCKT = 4,
  h_lambda,
  Lambda = identity,
  kernel.name = "Epa",
  Kfolds_lambda = 10,
  l_norm = 1,
  matrixSignsPairs = NULL,
  progressBars = "global",
  observedX1 = NULL,
  observedX2 = NULL,
  observedZ = NULL
)
```

Arguments

X1	a vector of n observations of the first variable X_1 .
X2	a vector of n observations of the second variable X_2 .
Z	a vector of n observations of the conditioning variable, or a matrix with n rows of observations of the conditioning vector (if Z is multivariate).
ZToEstimate	the new data of observations of Z at which the conditional Kendall's tau should be estimated.
designMatrixZ	the transformation of the ZToEstimate that will be used as predictors. By default, no transformation is applied.
typeEstCKT	type of estimation of the conditional Kendall's tau.
h_lambda	the smoothing bandwidth used in the cross-validation procedure to choose lambda.
Lambda	the function to be applied on conditional Kendall's tau. By default, the identity function is used.
kernel.name	name of the kernel. Possible choices are "Gaussian" (Gaussian kernel) and "Epa" (Epanechnikov kernel).
Kfolds_lambda	the number of folds used in the cross-validation procedure to choose lambda.
l_norm	type of norm used for selection of the optimal lambda. l_norm=1 corresponds to the sum of absolute values of differences between predicted and estimated conditional Kendall's tau while l_norm=2 corresponds to the sum of squares of differences.
matrixSignsPairs	the results of a call to computeMatrixSignPairs (if already computed). If NULL (the default value), the matrixSignsPairs will be computed again from the data.
progressBars	should progress bars be displayed? Possible values are <ul style="list-style-type: none"> • "none": no progress bar at all. • "global": only one global progress bar (default behavior) • "eachStep": uses a global progress bar + one progress bar for each kernel smoothing step.
observedX1, observedX2, observedZ	old parameter names for X1, X2, Z. Support for this will be removed at a later version.

Value

A list with the following components

- lambdaCV: the chosen value of the penalization parameters lambda.
- vectorLambda: a vector containing the values of lambda that have been compared.
- vectorMSEMean: the estimated MSE for each value of lambda in vectorLambda
- vectorMSESD: the estimated standard deviation of the MSE for each lambda. It can be used to construct confidence intervals for estimates of the MSE given by vectorMSEMean.

References

Derumigny, A., & Fermanian, J. D. (2020). On Kendall's regression. *Journal of Multivariate Analysis*, 178, 104610.

See Also

the main fitting function [CKT.kendallReg.fit](#).

Examples

```
# We simulate from a conditional copula
set.seed(1)
N = 400
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = -0.9 + 1.8 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

newZ = seq(2, 10, by = 0.1)
result <- CKT.KendallReg.LambdaCV(X1 = X1, X2 = X2, Z = Z,
  ZToEstimate = newZ, h_lambda = 2)

plot(x = result$vectorLambda, y = result$vectorMSEMean,
  type = "l", log = "x")
```

CKT.kernel

Estimation of conditional Kendall's tau using kernel smoothing

Description

Let X_1 and X_2 be two random variables. The goal of this function is to estimate the conditional Kendall's tau (a dependence measure) between X_1 and X_2 given $Z = z$ for a conditioning variable Z . Conditional Kendall's tau between X_1 and X_2 given $Z = z$ is defined as:

$$P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) > 0 | Z_1 = Z_2 = z)$$

$$-P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) < 0 | Z_1 = Z_2 = z),$$

where $(X_{1,1}, X_{1,2}, Z_1)$ and $(X_{2,1}, X_{2,2}, Z_2)$ are two independent and identically distributed copies of (X_1, X_2, Z) . For this, a kernel-based estimator is used, as described in (Derumigny, & Fermanian (2019)).

Usage

```

CKT.kernel(
  X1 = NULL,
  X2 = NULL,
  Z = NULL,
  newZ,
  h,
  kernel.name = "Epa",
  se = FALSE,
  confint = FALSE,
  level = 0.95,
  methodCV = "Kfolds",
  Kfolds = 5,
  nPairs = NULL,
  typeEstCKT = "wdm",
  progressBar = 1,
  warnNA = TRUE,
  observedX1 = NULL,
  observedX2 = NULL,
  observedZ = NULL
)

```

Arguments

X1	a vector of n observations of the first variable (or a 1-column matrix)
X2	a vector of n observations of the second variable (or a 1-column matrix)
Z	a vector of n observations of the conditioning variable, or a matrix with n rows of observations of the conditioning vector (in the case that several conditioning variables are given; in this case, each column corresponds to 1 conditioning variable). It can also be a data.frame, provided that all the entries are numeric.
newZ	the new data of observations of Z at which the conditional Kendall's tau should be estimated. It must have the same number of column as Z. It can be a vector (if NCOL(Z) == 1), a matrix or a data.frame, provided that all the entries are numeric.
h	the bandwidth used for kernel smoothing. If this is a vector, then cross-validation is used following the method given by argument methodCV to choose the best bandwidth before doing the estimation.
kernel.name	name of the kernel used for smoothing. Possible choices are "Gaussian" (Gaussian kernel) and "Epa" (Epanechnikov kernel).
se, confint	if TRUE, compute (asymptotic) standard errors and confidence intervals of conditional Kendall's tau. Standard errors are computed using Proposition 9 of (Derumigny & Fermanian, 2019).
level	the confidence level for the confidence intervals. By default, 95% confidence intervals are computed, i.e. level = 0.95.
methodCV	method used for the cross-validation. Possible choices are "leave-one-out" and "Kfolds".

Kfolds	number of subsamples used, if methodCV = "Kfolds".
nPairs	number of pairs used in the cross-validation criteria, if methodCV = "leave-one-out". Use nPairs = "all" to choose all pairs. The default is nPairs = $10 * n$, where n is the sample size.
typeEstCKT	type of estimation of the conditional Kendall's tau. Possible choices are <ul style="list-style-type: none"> • 1 and 3 produced biased estimators. 2 does not attain the full range $[-1, 1]$. Therefore these 3 choices are not recommended for applications on real data. • 4 is an improved version of 1, 2, 3 that has less bias and attains the full range $[-1, 1]$. • "wdm" is the default version and produces the same results as 4 when they are no ties in the data.
progressBar	control the display of progress bars. Possible choices are: <ul style="list-style-type: none"> • 0 no progress bar is displayed • 1 a general progress bar is displayed • 2 and larger values: a general progress bar is displayed, and additionally, a progressbar for each value of h is displayed to show the progress of the computation. This only applies when the bandwidth is chosen by cross-validation (i.e. when h is a vector).
warnNA	a Boolean to indicate whether warnings should be raised if NAs are produced. By default it is TRUE. If warnNA=FALSE, then no warning is raised even if NAs are produced. This is the case usually if either the bandwidth h is too small, or if there are already NAs in (some of) the inputs.
observedX1, observedX2, observedZ	old parameter names for X1, X2, Z. Support for this will be removed at a later version.

Details

Choice of the bandwidth h . The choice of the bandwidth must be done carefully. In the univariate case, the default kernel (Epanechnikov kernel) has a support on $[-1, 1]$, so for a bandwidth h , estimation of conditional Kendall's tau at $Z = z$ will only use points for which $Z_i \in [z \pm h]$. As usual in nonparametric estimation, h should not be too small (to avoid having a too large variance) and should not be large (to avoid having a too large bias).

We recommend that for each z for which the conditional Kendall's tau $\tau_{X_1, X_2 | Z=z}$ is estimated, the set $\{i : Z_i \in [z \pm h]\}$ should contain at least 20 points and not more than 30% of the points of the whole dataset. Note that for a consistent estimation, as the sample size n tends to the infinity, h should tend to 0 while the size of the set $\{i : Z_i \in [z \pm h]\}$ should also tend to the infinity. Indeed the conditioning points should be closer and closer to the point of interest z (small h) and more and more numerous (h tending to 0 slowly enough).

In the multivariate case, similar recommendations can be made. Because of the curse of dimensionality, a larger sample will be necessary to reach the same level of precision as in the univariate case.

Value

an S3 object of class `estimated_CKT_kernel` with components including:

- `estimatedCKT` the vector of size `NROW(newZ)` containing the values of the estimated conditional Kendall's tau.
- `finalh` the bandwidth `h` that was finally used for kernel smoothing (either the one specified by the user or the one chosen by cross-validation if multiple bandwidths were given.)
- `resultCV` (only in case of cross-validation). This gives the output of the cross-validation function that is used, i.e. the output of either `CKT.hcv.11out` or `CKT.hcv.Kfolds`.
- `se`, and `confint` if requested.

Some methods (`se`, `confint` and `plot`) are available for such an object, see `plot.estimated_CKT_kernel`.

References

Derumigny, A., & Fermanian, J. D. (2019). On kernel-based estimation of conditional Kendall's tau: finite-distance bounds and asymptotic behavior. *Dependence Modeling*, 7(1), 292-321. doi:10.1515/demo20190016

See Also

`CKT.estimate` for other estimators of conditional Kendall's tau. `CKTmatrix.kernel` for a generalization of this function when the conditioned vector is of dimension `d` instead of dimension 2 here.

See `CKT.hcv.11out` for manual selection of the bandwidth `h` by leave-one-out or K-folds cross-validation.

Examples

```
# We simulate from a conditional copula
set.seed(1)
N = 100
# This is a small example for performance reason.
# For a better example, use:
# N = 800
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = -0.9 + 1.8 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

newZ = seq(2,10,by = 0.1)
estimatedCKT_kernel <- CKT.kernel(
  X1 = X1, X2 = X2, Z = Z,
  newZ = newZ, h = 0.1, kernel.name = "Epa")$estimatedCKT

# Comparison between true Kendall's tau (in black)
# and estimated Kendall's tau (in red)
trueConditionalTau = -0.9 + 1.8 * pnorm(newZ, mean = 5, sd = 2)
```

```

plot(newZ, trueConditionalTau , col = "black",
     type = "l", ylim = c(-1, 1))
lines(newZ, estimatedCKT_kernel, col = "red")

# Multivariate example
N = 100
# This is a small example for performance reason.
# For a better example, use:
# N = 1000
Z1 = rnorm(n = N, mean = 5, sd = 2)
Z2 = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = -0.9 + 1.8 * pnorm(Z1 - Z2, mean = 2, sd = 2)
simCopula = VineCopula::BiCopSim(N = N , family = 1,
    par = VineCopula::BiCopTau2Par(1 , conditionalTau ))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

Z = cbind(Z1, Z2)

newZ = expand.grid(Z1 = seq(2,8,by = 0.5),
                  Z2 = seq(2,8,by = 1))
estimatedCKT_kernel <- CKT.kernel(
  X1 = X1, X2 = X2, Z = Z,
  newZ = newZ, h = 1, kernel.name = "Epa")$estimatedCKT

if (requireNamespace("ggplot2", quietly = TRUE)) {
  df = rbind(
    data.frame(newZ, CKT = estimatedCKT_kernel,
               type = "estimated CKT") ,
    data.frame(newZ, CKT = -0.9 + 1.8 * pnorm(newZ$Z1 - newZ$Z2,
                                             mean = 2, sd = 2),
               type = "true CKT")
  )

  ggplot2::ggplot(df) +
    ggplot2::geom_tile(ggplot2::aes(x = Z1, y = Z2, fill = CKT)) +
    ggplot2::facet_grid(as.formula("~type"))
}

```

CKT.predict.kNN

Prediction of conditional Kendall's tau using nearest neighbors

Description

Let X_1 and X_2 be two random variables. The goal of this function is to estimate the conditional Kendall's tau (a dependence measure) between X_1 and X_2 given $Z = z$ for a conditioning variable Z . Conditional Kendall's tau between X_1 and X_2 given $Z = z$ is defined as:

$$P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) > 0 | Z_1 = Z_2 = z)$$

$$-P((X_{1,1} - X_{2,1})(X_{1,2} - X_{2,2}) < 0 | Z_1 = Z_2 = z),$$

where $(X_{1,1}, X_{1,2}, Z_1)$ and $(X_{2,1}, X_{2,2}, Z_2)$ are two independent and identically distributed copies of (X_1, X_2, Z) . In other words, conditional Kendall's tau is the difference between the probabilities of observing concordant and discordant pairs from the conditional law of

$$(X_1, X_2) | Z = z.$$

This function estimates conditional Kendall's tau using a **nearest neighbors**. This is possible by the relationship between estimation of conditional Kendall's tau and classification problems (see Derumigny and Fermanian (2019)): estimation of conditional Kendall's tau is equivalent to the prediction of concordance in the space of pairs of observations.

Usage

```
CKT.predict.kNN(
  datasetPairs,
  designMatrix = datasetPairs[, 2:(ncol(datasetPairs) - 3), drop = FALSE],
  newZ,
  number_nn,
  weightsVariables = 1,
  normLp = 2,
  constantA = 1,
  partition = NULL,
  verbose = 1,
  lengthVerbose = 100,
  methodSort = "partial.sort"
)
```

Arguments

datasetPairs	the matrix of pairs and corresponding values of the kernel as provided by datasetPairs .
designMatrix	the matrix of predictors. They must have the same number of variables as newZ and the same number of observations as inputMatrix, i.e. there should be one "multivariate observation" of the predictor for each pair.
newZ	the matrix of predictors for which we want to estimate the conditional Kendall's taus at these values.
number_nn	vector of numbers of nearest neighbors to use. If several number of neighbors are given (local) aggregation is performed using Lepski's method on the subset determined by the partition.
weightsVariables	optional argument to give different weights w_j to each variable.
normLp	the p in the weighted p-norm $\ x\ _p = \sum_j w_j * x_j^p$ used to determine the distance in the computation of the nearest neighbors.
constantA	a tuning parameter that controls the adaptation. The higher, the smoother it is; while the smaller, the least smooth it is.
partition	used only if <code>length(number_nn) > 1</code> . It is the number of subsets to consider for the local choice of the number of nearest neighbors ; or a vector giving the id of each observations among the subsets. If NULL, only one set is used.

verbose if TRUE, this print information each lengthVerbose iterations
lengthVerbose number of iterations at each time for which progress is printed.
methodSort is the sorting method used to find the nearest neighbors. Possible choices are `ecdf` (uses the `ecdf` to order the points to find the neighbors) and `partial.sort` uses a partial sorting algorithm. This parameter should not matter except for the computation time.

Value

a list with two components

- `estimatedCKT` the estimated conditional Kendall's tau, a vector of the same size as the number of rows in `newZ`;
- `vect_k_chosen` the locally selected number of nearest neighbors, a vector of the same size as the number of rows in `newZ`.

References

Derumigny, A., & Fermanian, J. D. (2019). A classification point-of-view about conditional Kendall's tau. *Computational Statistics & Data Analysis*, 135, 70-94. (Algorithm 5) [doi:10.1016/j.csda.2019.01.013](https://doi.org/10.1016/j.csda.2019.01.013)

See Also

See also other estimators of conditional Kendall's tau: [CKT.fit.tree](#), [CKT.fit.randomForest](#), [CKT.fit.nNets](#), [CKT.fit.randomForest](#), [CKT.fit.GLM](#), [CKT.kernel](#), [CKT.kendallReg.fit](#), and the more general wrapper [CKT.estimate](#).

Examples

```
# We simulate from a conditional copula
set.seed(1)
N = 800
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = -0.9 + 1.8 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

newZ = seq(2,10,by = 0.1)
datasetP = datasetPairs(X1 = X1, X2 = X2, Z = Z, h = 0.07, cut = 0.9)
estimatedCKT_knn <- CKT.predict.kNN(
  datasetPairs = datasetP,
  newZ = matrix(newZ, ncol = 1),
  number_nn = c(50,80, 100, 120,200),
  partition = 8)

# Comparison between true Kendall's tau (in black)
# and estimated Kendall's tau (in red)
trueConditionalTau = -0.9 + 1.8 * pnorm(newZ, mean = 5, sd = 2)
plot(newZ, trueConditionalTau, col="black",
```

```
type = "l", ylim = c(-1, 1))
lines(newZ, estimatedCKT_knn$estimatedCKT, col = "red")
```

CKT.predict.nNets	<i>Predict the values of conditional Kendall's tau using Model Averaging of Neural Networks</i>
-------------------	---

Description

Predict the values of conditional Kendall's tau using Model Averaging of Neural Networks

Usage

```
CKT.predict.nNets(fit, newZ, aggregationMethod = "mean")
```

Arguments

fit	result of a call to <code>CKT.fit.nNet</code>
newZ	new matrix of observations, with the same number of variables. and same names as the <code>designMatrix</code> that was used to fit the neural networks.
aggregationMethod	the method to be used to aggregate all the predictions together. Can be "mean" or "median".

Value

CKT.predict.nNets returns a vector of (predicted) conditional Kendall's taus of the same size as the number of rows of the matrix `newZ`.

CKTmatrix.kernel	<i>Estimate the conditional Kendall's tau matrix at different conditioning points</i>
------------------	---

Description

Assume that we are interested in a random vector (X, Z) , where X is of dimension $d > 2$ and Z is of dimension 1. We want to estimate the dependence across the elements of the conditioned vector X given $Z = z$. This function takes in parameter observations of (X, Z) and returns kernel-based estimators of

$$\tau_{i,j|Z=z_k}$$

which is the conditional Kendall's tau between X_i and X_j given to $Z = z_k$, for every conditioning point z_k in `gridZ`. If the conditional Kendall's tau matrix has a block structure, then improved estimation is possible by averaging over the kernel-based estimators of pairwise conditional Kendall's taus. Groups of variables composing the same blocks can be defined using the parameter `blockStructure`, and the averaging can be set on using the parameter `averaging=all`, or `averaging=diag` for faster estimation by averaging only over diagonal elements of each block.

Usage

```

CKTmatrix.kernel(
  dataMatrix,
  observedZ,
  gridZ,
  averaging = "no",
  blockStructure = NULL,
  h,
  kernel.name = "Epa",
  typeEstCKT = "wdm"
)

```

Arguments

dataMatrix	a matrix of size (n,d) containing n observations of a d-dimensional random vector X .
observedZ	vector of observed points of a conditioning variable Z . It must have the same length as the number of rows of dataMatrix.
gridZ	points at which the conditional Kendall's tau is computed.
averaging	type of averaging used for fast estimation. Possible choices are <ul style="list-style-type: none"> • no: no averaging; • all: averaging all Kendall's taus in each block; • diag: averaging along diagonal blocks elements.
blockStructure	list of vectors. Each vector corresponds to one group of variables and contains the indexes of the variables that belongs to this group. blockStructure must be a partition of $1:d$, where d is the number of columns in dataMatrix.
h	bandwidth. It can be a real, in this case the same h will be used for every element of gridZ. If h is a vector then its elements are recycled to match the length of gridZ.
kernel.name	name of the kernel used for smoothing. Possible choices are: "Gaussian" (Gaussian kernel) and "Epa" (Epanechnikov kernel).
typeEstCKT	type of estimation of the conditional Kendall's tau.

Value

array with dimensions depending on averaging:

- If averaging = "no": it returns an array of dimensions (n, n, length(gridZ)), containing the estimated conditional Kendall's tau matrix given $Z = z$. Here, n is the number of rows in dataMatrix.
- If averaging = "all" or "diag": it returns an array of dimensions (length(blockStructure), length(blockStructure), length(gridZ)), containing the block estimates of the conditional Kendall's tau given $Z = z$ with ones on the diagonal.

Author(s)

Rutger van der Spek, Alexis Derumigny

References

van der Spek, R., & Derumigny, A. (2025). Fast estimation of Kendall's Tau and conditional Kendall's Tau matrices under structural assumptions. [doi:10.1515/demo20250012](https://doi.org/10.1515/demo20250012)

See Also

[CKT.kernel](#) for kernel-based estimation of conditional Kendall's tau between two variables (i.e. the equivalent of this function when X is bivariate and $d=2$).

Examples

```
# Data simulation
n = 100
Z = runif(n)
d = 5
CKT_11 = 0.8
CKT_22 = 0.9
CKT_12 = 0.1 + 0.5 * cos(pi * Z)
data_X = matrix(nrow = n, ncol = d)
for (i in 1:n){
  CKT_matrix = matrix(data =
    c( 1      , CKT_11  , CKT_11  , CKT_12[i], CKT_12[i] ,
      CKT_11  , 1      , CKT_11  , CKT_12[i], CKT_12[i] ,
      CKT_11  , CKT_11  , 1      , CKT_12[i], CKT_12[i] ,
      CKT_12[i], CKT_12[i], CKT_12[i], 1      , CKT_22  ,
      CKT_12[i], CKT_12[i], CKT_12[i], CKT_22  , 1      ),
    nrow = 5, ncol = 5)
  sigma = sin(pi * CKT_matrix/2)
  data_X[i, ] = mvtnorm::rmvnorm(n = 1, sigma = sigma)
}
plot(as.data.frame.matrix(data_X))

# Estimation of CKT matrix
h = 1.06 * sd(Z) * n^{-1/5}
gridZ = c(0.2, 0.8)
estMatrixAll <- CKTmatrix.kernel(
  dataMatrix = data_X, observedZ = Z, gridZ = gridZ, h = h)
# Averaging estimator
estMatrixAve <- CKTmatrix.kernel(
  dataMatrix = data_X, observedZ = Z, gridZ = gridZ,
  averaging = "diag", blockStructure = list(1:3,4:5), h = h)

# The estimated CKT matrix conditionally to Z=0.2 is:
estMatrixAll[ , , 1]
# Using the averaging estimator,
# the estimated CKT between the first group (variables 1 to 3)
# and the second group (variables 4 and 5) is
estMatrixAve[1, 2, 1]

# True value (of CKT between variables in block 1 and 2 given Z = 0.2):
0.1 + 0.5 * cos(pi * 0.2)
```

computeKernelMatrix	<i>Computing the kernel matrix</i>
---------------------	------------------------------------

Description

This function computes a matrix of dimensions $(\text{length}(\text{observedX3}), \text{length}(\text{newX3}))$, whose element at coordinate (i, j) is $K_h(\text{observedX3}[i] - \text{newX3}[j])$, where $K_h(x) := K(x/h)/h$ and K is the kernel.

Usage

```
computeKernelMatrix(observedX, newX, kernel, h)
```

Arguments

observedX	a numeric vector of observations of X3. on the interval $[0, 1]$.
newX	a numeric vector of points of X3.
kernel	a character string describing the kernel to be used. Possible choices are Gaussian, Triangular and Epanechnikov.
h	the bandwidth

Value

a numeric matrix of dimensions $(\text{length}(\text{observedX}), \text{length}(\text{newX}))$

See Also

[estimateCondCDF_matrix](#), [estimateCondCDF_vec](#),

Examples

```
Y = MASS::mvrnorm(n = 100, mu = c(0,0), Sigma = cbind(c(1, 0.9), c(0.9, 1)))
matrixK = computeKernelMatrix(observedX = Y[,2], newX = c(0, 1, 2.5),
kernel = "Gaussian", h = 0.8)

# To have an estimator of the conditional expectation of Y1 given Y2 = 0, 1, 2.5
Y[,1] * matrixK[,1] / sum(matrixK[,1])
Y[,1] * matrixK[,2] / sum(matrixK[,2])
Y[,1] * matrixK[,3] / sum(matrixK[,3])
```

computeMatrixSignPairs

Compute the matrix of signs of pairs

Description

Compute a matrix giving the concordance or discordance of each pair of observations.

Usage

```
computeMatrixSignPairs(vectorX1, vectorX2, typeEstCKT = 4)
```

Arguments

vectorX1 vector of observed data (first coordinate)

vectorX2 vector of observed data (second coordinate)

typeEstCKT if typeEstCKT = 2 or 4, compute the matrix whose term (i,j) is :

$$1\{(X_{i,1} - X_{j,1}) * (X_{i,2} - X_{j,2}) > 0\} - 1\{(X_{i,1} - X_{j,1}) * (X_{i,2} - X_{j,2}) < 0\},$$

where 1 is the indicator function.

For typeEstCKT = 1 (respectively typeEstCKT = 3) a negatively biased (respectively positively) matrix is given.

Value

an $n * n$ matrix with the signs of each pair of observations.

Examples

```
# We simulate from a conditional copula
N = 500
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = 0.9 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N = N , family = 3,
  par = VineCopula::BiCopTau2Par(1 , conditionalTau) )
matrixPairs = computeMatrixSignPairs(vectorX1 = simCopula[,1],
  vectorX2 = simCopula[,2])
```

conv_treeCKT	<i>Converting to matrix of indicators / matrix of conditional Kendall's tau</i>
--------------	---

Description

The function `treeCKT2matrixInd` takes as input a binary tree that has been returned by the function `bCond.treeCKT`. Since this tree describes a partition of the conditioning space, it can be interesting to get, for a given dataset, the matrix

$$1\{X_{i,J} \in A_{j,J}\},$$

where each $A_{j,J}$ corresponds to a conditioning subset. This is the so-called `matrixInd`. Finally, it can be interesting to get the matrix of

Usage

```
treeCKT2matrixInd(estimatedTree, newDataXJ = NULL)

matrixInd2matrixCKT(matrixInd, newDataXI)

treeCKT2matrixCKT(estimatedTree, newDataXI = NULL, newDataXJ = NULL)
```

Arguments

<code>estimatedTree</code>	the tree that has been estimated before, for example by <code>bCond.treeCKT</code> .
<code>newDataXJ</code>	this is a matrix of size $N * J $ where $ J $ is the number of conditional variables used in the tree. By default this is <code>NULL</code> meaning that we return the matrix for the original data (that was used to compute the <code>estimatedTree</code>).
<code>matrixInd</code>	a matrix of indexes of size $(n, N.\text{boxes})$ describing for each observation i to which box (= event) it belongs.
<code>newDataXI</code>	this is a matrix of size $N * I $ where $ I $ is the number of conditioned variables. By default this is <code>NULL</code> meaning that we return the matrix for the original data used to compute the <code>estimatedTree</code>

Value

- The function `treeCKT2matrixInd` returns a matrix of size $N * m$ which component $[i, j]$ is

$$1\{X_{i,J} \in A_{j,J}\}$$

- The function `matrixInd2matrixCKT` and `treeCKT2matrixCKT` return a matrix of size $|I| * (|I|-1) * m$ where each component corresponds to a conditional Kendall's tau between a pair of conditional variables conditionally to the conditioned variables in one of the boxes

See Also

[bCond.treeCKT](#) for the construction of such a binary tree.

Examples

```
set.seed(1)
n = 200
XJ = MASS::mvrnorm(n = n, mu = c(3,3), Sigma = rbind(c(1, 0.2), c(0.2, 1)))
XI = matrix(nrow = n, ncol = 2)
high_XJ1 = which(XJ[,1] > 4)
XI[high_XJ1, ] = MASS::mvrnorm(n = length(high_XJ1), mu = c(10,10),
                               Sigma = rbind(c(1, 0.8), c(0.8, 1)))
XI[-high_XJ1, ] = MASS::mvrnorm(n = n - length(high_XJ1), mu = c(8,8),
                               Sigma = rbind(c(1, -0.2), c(-0.2, 1)))

result = bCond.treeCKT(XI = XI, XJ = XJ, minSize = 10, verbose = 2)

treeCKT2matrixInd(result)

matrixInd2matrixCKT(treeCKT2matrixInd(result), newDataXI = XI)

treeCKT2matrixCKT(result)
```

datasetPairs	<i>Construct a dataset of pairs of observations for the estimation of conditional Kendall's tau</i>
--------------	---

Description

In (Derumigny, & Fermanian (2019)), it is described how the problem of estimating conditional Kendall's tau can be rewritten as a classification task for a dataset of pairs (of observations). This function computes such a dataset, that can be then used to estimate conditional Kendall's tau using one of the following functions: [CKT.fit.tree](#), [CKT.fit.randomForest](#), [CKT.fit.GLM](#), [CKT.fit.nNets](#), [CKT.predict.kNN](#).

Usage

```
datasetPairs(
  X1,
  X2,
  Z,
  h,
  cut = 0.9,
  onlyConsecutivePairs = FALSE,
  nPairs = NULL
)
```

Arguments

X1	vector of observations of the first conditioned variable.
X2	vector of observations of the second conditioned variable.
Z	vector or matrix of observations of the conditioning variable(s), of dimension dimZ.
h	the bandwidth. Can be a vector; in this case, the components of h will be reused to match the dimension of Z.
cut	the cutting level to keep a given pair or not. Used only if no nPairs is provided.
onlyConsecutivePairs	if TRUE, only consecutive pairs are used.
nPairs	number of most relevant pairs to keep in the final datasets. If this is different than the default NULL, the cutting level cut is not used.

Value

A matrix with $(4 + \text{dimZ})$ columns and $n \cdot (n-1) / 2$ rows if onlyConsecutivePairs=FALSE and else $(n/2)$ rows. It is structured in the following way:

- column 1 contains the information about the concordance of the pair (i,j) ;
- columns 2 to $1 + \text{dimZ}$ contain the mean value of Z (the conditioning variables) ;
- column $2 + \text{dimZ}$ contains the value of the kernel $K_h(Z_j - Z_i)$;
- column $3 + \text{dimZ}$ and $4 + \text{dimZ}$ contain the corresponding values of i and j.

References

Derumigny, A., & Fermanian, J. D. (2019). A classification point-of-view about conditional Kendall's tau. Computational Statistics & Data Analysis, 135, 70-94. (Algorithm 1 for all pairs and Algorithm 8 for the case of only consecutive pairs) [doi:10.1016/j.csda.2019.01.013](https://doi.org/10.1016/j.csda.2019.01.013)

See Also

the functions that require such a dataset of pairs to do the estimation of conditional Kendall's tau: [CKT.fit.tree](#), [CKT.fit.randomForest](#), [CKT.fit.GLM](#), [CKT.fit.nNets](#), [CKT.predict.kNN](#), and [CKT.fit.randomForest](#).

Examples

```
# We simulate from a conditional copula
N = 500
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = 0.9 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N = N , family = 3,
  par = VineCopula::BiCopTau2Par(1 , conditionalTau) )
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

datasetP = datasetPairs(
  X1 = X1, X2 = X2, Z = Z, h = 0.07, cut = 0.9)
```

estimateCondCDF_matrix

Compute kernel-based conditional marginal (univariate) cdfs

Description

This function computes an estimate of the conditional (marginal) cdf of X1 given a conditioning variable X3.

Usage

```
estimateCondCDF_matrix(observedX1, newX1, matrixK3)
```

Arguments

observedX1 a sample of observations of X1 of size n
newX1 a sample of new points for the variable X1, of size p1
matrixK3 a matrix of kernel values of dimension (p3, n) $(K_h(X3[i] - U3[j]))_{i,j}$ such as given by [computeKernelMatrix](#).

Details

This function is supposed to be used with [computeKernelMatrix](#). Assume that we observe a sample $(X_{i,1}, X_{i,3}), i = 1, \dots, n$. We want to estimate the conditional cdf of X_1 given $X_3 = x_3$ at point x_1 using the following kernel-based estimator

$$\hat{P}(X_1 \leq x_1 | X_3 = x_3) := \frac{\sum_{l=1}^n 1\{X_{l,1} \leq x_1\} K_h(X_{l,3} - x_3)}{\sum_{l=1}^n K_h(X_{l,3} - x_3)},$$

for every x_1 in newX1 and every x_3 in newX3. The matrixK3 should be a matrix of the values $K_h(X_{l,3} - x_3)$ such as the one produced by [computeKernelMatrix](#)(observedX3, newX3, kernel, h).

Value

A matrix of dimensions (p1 = length(newX), p3 = length(matrixK3[,1])) of estimators $\hat{P}(X_1 \leq x_1 | X_3 = x_3)$ for every possible choices of (x_1, x_3) .

Examples

```
Y = MASS::mvrnorm(n = 100, mu = c(0,0), Sigma = cbind(c(1, 0.9), c(0.9, 1)))
newY1 = seq(-1, 1, by = 0.5)
newY2 = c(0, 1, 2)
matrixK = computeKernelMatrix(observedX = Y[,2], newX = newY2,
  kernel = "Gaussian", h = 0.8)
# In this matrix, there are the estimated conditionl cdf at points given by newY1
# conditionally to the points given by newY2.
matrixCondCDF = estimateCondCDF_matrix(observedX1 = Y[,1],
```

```
newX1 = newY1, matrixK)
matrixCondCDF
```

estimateCondCDF_vec	<i>Compute kernel-based conditional marginal (univariate) cdfs</i>
---------------------	--

Description

This function computes an estimate of the conditional (marginal) cdf of X1 given a conditioning variable X3. This function is supposed to be used with [computeKernelMatrix](#). Assume that we observe a sample $(X_{i,1}, X_{i,3}), i = 1, \dots, n$. We want to estimate the conditional cdf of X_1 given $X_3 = x_3$ at point x_1 using the following kernel-based estimator

$$\hat{P}(X_1 \leq x_1 | X_3 = x_3) := \frac{\sum_{l=1}^n 1\{X_{l,1} \leq x_1\} K_h(X_{l,3} - x_3)}{\sum_{l=1}^n K_h(X_{l,3} - x_3)},$$

for every couple $(x_{j,1}, x_{j,3})$ where $x_{j,1}$ in newX1 and $x_{j,3}$ in newX3. The matrixK3 should be a matrix of the values $K_h(X_{l,3} - x_3)$ such as the one produced by [computeKernelMatrix](#)(observedX3, newX3, kernel, h).

Usage

```
estimateCondCDF_vec(observedX1, newX1, matrixK3)
```

Arguments

observedX1	a sample of observations of X1 of size n
newX1	a sample of new points for the variable X1, of size p1
matrixK3	a matrix of kernel values of dimension (p2 , n) $(K_h(X3[i] - U3[j]))_{i,j}$ such as given by computeKernelMatrix .

Value

It returns a vector of length newX1 of estimators $\hat{P}(X_1 \leq x_1 | X_3 = x_3)$ for every couple $(x_{j,1}, x_{j,3})$.

Examples

```
Y = MASS::mvrnorm(n = 100, mu = c(0,0), Sigma = cbind(c(1, 0.9), c(0.9, 1)))
newY1 = seq(-1, 1, by = 0.5)
newY2 = newY1
matrixK = computeKernelMatrix(observedX = Y[,2], newX = newY2,
  kernel = "Gaussian", h = 0.8)
vecCondCDF = estimateCondCDF_vec(observedX1 = Y[,1],
  newX1 = newY1, matrixK)
vecCondCDF
```

estimateCondQuantiles *Compute kernel-based conditional quantiles*

Description

This function is supposed to be used with `computeKernelMatrix`. Assume that we observe a sample $(X_{i,1}, X_{i,3}), i = 1, \dots, n$. We want to estimate the conditional quantiles of X_1 given $X_3 = x_3$ at point u_1 using the following kernel-based estimator

$$\hat{Q}(u_1|X_3 = x_3) := \hat{P}^{(-1)}(u_1 \leq x_1|X_3 = x_3),$$

where

$$\hat{P}(X_1 \leq x_1|X_3 = x_3) := \frac{\sum_{l=1}^n 1\{X(l,1) \leq x_1\} K_h(X(l,3) - x_3)}{\sum_{l=1}^n K_h(X(l,3) - x_3)},$$

for every u_1 in `probsX1` and every x_3 in `newX3`. The `matrixK3` should be a matrix of the values $K_h(X(l,3) - x_3)$ such as the one produced by `computeKernelMatrix`(`observedX3`, `newX3`, `kernel`, `h`).

Usage

```
estimateCondQuantiles(observedX1, probsX1, matrixK3)
```

Arguments

<code>observedX1</code>	a sample of observations of X1 of size n
<code>probsX1</code>	a sample of probabilities at which we want to compute the quantiles for the variable X1, of size p1
<code>matrixK3</code>	a matrix of kernel values of dimension (p2 , n) $(K_h(X3[i] - U3[j]))_{i,j}$ such as given by <code>computeKernelMatrix</code> .

Value

A matrix of dimensions (p1,p2) whose (i,j) entry is $\hat{Q}(u_1|X_3 = x_3)$ with $u_1 = \text{probsX1}[i]$ and $x_3 = \text{newX3}[j]$, where `newX3[j]` is the vector that was used to construct `matrixK3`.

Examples

```
Y = MASS::mvrnorm(n = 100, mu = c(0,0), Sigma = cbind(c(1, 0.9), c(0.9, 1)))
matrixK = computeKernelMatrix(observedX = Y[,2] , newX = c(0, 1, 2.5),
  kernel = "Gaussian", h = 0.8)
matrixnp = estimateCondQuantiles(observedX1 = Y[,2],
  probsX1 = c(0.3, 0.5) , matrixK3 = matrixK)
matrixnp
```

estimateNPCondCopula *Compute a kernel-based estimator of the conditional copula*

Description

Assuming that we observe a sample $(X_{i,1}, X_{i,2}, X_{i,3}), i = 1, \dots, n$, this function returns a array $\hat{C}_{1,2|3}(u_1, u_2 | X_3 = x_3)$ for each choice of (u_1, u_2, x_3).

Usage

```
estimateNPCondCopula(
  X1 = NULL,
  X2 = NULL,
  X3 = NULL,
  U1_,
  U2_,
  newX3,
  kernel,
  h,
  observedX1 = NULL,
  observedX2 = NULL,
  observedX3 = NULL
)
```

Arguments

X1, X2, X3	vectors of observations of size n
U1_	a vector of numbers in [0, 1]
U2_	a vector of numbers in [0, 1]
newX3	a vector of new values for the conditioning variable X3
kernel	a character string describing the kernel to be used. Possible choices are Gaussian, Triangular and Epanechnikov.
h	the bandwidth to use in the estimation.
observedX1, observedX2, observedX3	old parameter names for X1, X2, X3. Support for this will be removed at a later version.

Value

An array of dimension $(\text{length}(U1_, U2_, newX3))$ whose element in position (i, j, k) is $\hat{C}_{1,2|3}(u_1, u_2 | X_3 = x_3)$ where $u_1 = U1_[i]$, $u_2 = U2_[j]$ and $x_3 = newX3[k]$

References

Derumigny, A., & Fermanian, J. D. (2017). About tests of the “simplifying” assumption for conditional copulas. *Dependence Modeling*, 5(1), 154-197. doi:[10.1515/demo20170011](https://doi.org/10.1515/demo20170011)

See Also

[estimateParCondCopula](#) for estimating a conditional copula in a parametric setting (= where the conditional copula is assumed to belong to a parametric class). [simpA.NP](#) for a test that this conditional copula is constant with respect to the value x_3 of the conditioning variable.

Examples

```
# We simulate from a conditional copula
N = 500
X3 = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = 0.9 * pnorm(X3, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N, family = 3,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

# We do the estimation
grid = c(0.2, 0.4, 0.6, 0.8)
arrayEst = estimateNPCondCopula(
  X1 = X1, X2 = X2, X3 = X3,
  U1_ = grid, U2_ = grid, newX3 = c(2, 5, 7),
  kernel = "Gaussian", h = 0.8)
arrayEst
```

estimateParCondCopula *Estimation of parametric conditional copulas*

Description

The function `estimateParCondCopula` computes an estimate of the conditional parameters in a conditional parametric copula model, i.e.

$$C_{X_1, X_2 | X_3 = x_3} = C_{\theta(x_3)},$$

for some parametric family (C_θ), some conditional parameter $\theta(x_3)$, and a three-dimensional random vector (X_1, X_2, X_3) . Remember that $C_{X_1, X_2 | X_3 = x_3}$ denotes the conditional copula of X_1 and X_2 given $X_3 = x_3$.

The function `estimateParCondCopula_ZIJ` is an auxiliary function that is called when conditional pseudos-observations are already available when one wants to estimate a parametric conditional copula.

Usage

```
estimateParCondCopula(
  X1 = NULL,
  X2 = NULL,
  X3 = NULL,
```

```

    newX3,
    family,
    method = "mle",
    h,
    observedX1 = NULL,
    observedX2 = NULL,
    observedX3 = NULL
  )

estimateParCondCopula_ZIJ(Z1_J, Z2_J, observedX3, newX3, family, method, h)

```

Arguments

X1	a vector of n observations of the first conditioned variable
X2	a vector of n observations of the second conditioned variable
X3	a vector of n observations of the conditioning variable
newX3	a vector of new observations of X_3
family	an integer indicating the parametric family of copulas to be used, following the conventions of the VineCopula package, see e.g. VineCopula::BiCop .
method	the method of estimation of the conditional parameters. Can be "mle" for maximum likelihood estimation or "itau" for estimation by inversion of Kendall's tau.
h	bandwidth to be chosen
observedX1, observedX2, observedX3	old parameter names for X1, X2, X3. Support for this will be removed at a later version.
Z1_J	the conditional pseudos-observations of the first variable, i.e. $\hat{F}_{1 J}(x_{i,1} x_J = x_{i,J})$ for $i = 1, \dots, n$.
Z2_J	the conditional pseudos-observations of the second variable, i.e. $\hat{F}_{2 J}(x_{i,2} x_J = x_{i,J})$ for $i = 1, \dots, n$.

Value

a vector of size `length(newX3)` containing the estimated conditional copula parameters for each value of `newX3`.

References

Derumigny, A., & Fermanian, J. D. (2017). About tests of the “simplifying” assumption for conditional copulas. *Dependence Modeling*, 5(1), 154-197. doi:[10.1515/demo20170011](#)

See Also

[estimateNPCondCopula](#) for estimating a conditional copula in a nonparametric setting (= without parametric assumption on the conditional copula). [simpA.param](#) for a test that this conditional copula is constant with respect to the value x_3 of the conditioning variable.

Examples

```
# We simulate from a conditional copula
N = 500

X3 = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = 0.9 * pnorm(X3, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(
  N=N , family = 1, par = VineCopula::BiCopTau2Par(1 , conditionalTau ))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

gridnewX3 = seq(2, 8, by = 1)
conditionalTauNewX3 = 0.9 * pnorm(gridnewX3, mean = 5, sd = 2)

vecEstimatedThetas = estimateParCondCopula(
  X1 = X1, X2 = X2, X3 = X3,
  newX3 = gridnewX3, family = 1, h = 0.1)

# Estimated conditional parameters
vecEstimatedThetas
# True conditional parameters
VineCopula::BiCopTau2Par(1 , conditionalTauNewX3 )

# Estimated conditional Kendall's tau
VineCopula::BiCopPar2Tau(1 , vecEstimatedThetas )
# True conditional Kendall's tau
conditionalTauNewX3
```

measures_nonsimplifyingness_NP

Compute a measure of non-simplifyingness based on non-parametric estimation of the conditional copula

Description

Compute a measure of non-simplifyingness based on non-parametric estimation of the conditional copula

Usage

```
measures_nonsimplifyingness_NP(
  X1,
  X2,
  Z,
  h,
  measures = "all",
  kernel.name = "Epanechnikov",
```

```

truncVal = NULL,
numericalInt = list(kind = "legendre", nGrid = 10),
verbose = 0
)

```

Arguments

X1, X2	vector of n observations of the conditioned variables
Z	vector of n observations of the conditioning variable
h	the bandwidth used for kernel smoothing
measures	choices of measures of non-simplifyingness to be computed. <code>measures = "all"</code> includes all available non-parametric measures of non-simplifyingness. Otherwise, measures must be a character vector and a subset of <code>c("T1_CvM-Cs3", "T1_CvM-Cs4", "tilde_T0_CvM", "T1_KS-Cs3", "T1_KS-Cs4", "tilde_T0_KS")</code> .
kernel.name	the name of the kernel
truncVal	the value of truncation for the integral, i.e. the integrals are computed from <code>truncVal</code> to <code>1-truncVal</code> instead of from 0 to 1. Note that <code>truncVal</code> must be in the interval $[0, 0.5)$, i.e. 0 is allowed but not 0.5. The default is <code>truncVal = NULL</code> , which actually means that <code>truncVal = h</code> if $h < 0.5$ and <code>truncVal = 0</code> else.
numericalInt	parameters to be given to <code>statmod::gauss.quad</code> , including the number of quadrature points and the type of interpolation.
verbose	option used for debugging. If <code>verbose = 0</code> , the function is silent. Higher values of <code>verbose</code> give more explicit details on the computations.

Value

a data.frame where each row corresponds to one measure of non-simplifyingness and one choice of `h`. As a particular case, if `measures` and `h` are both of length 1, this data.frame will have only one row.

References

Derumigny, A. (2025). Measures of non-simplifyingness for conditional copulas and vines. ArXiv preprint, arXiv:2504.07704. [doi:10.48550/arXiv.2504.07704](https://doi.org/10.48550/arXiv.2504.07704)

See Also

[simpA.NP\(\)](#) for non-parametric tests of the simplifying assumption.

Examples

```

set.seed(1)
N = 500
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = 0.8
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))

```

```

X1 = qnorm(simCopula[,1], mean = Z)
X2 = qnorm(simCopula[,2], mean = - Z)

result <- measures_nonsimplifyingness_NP(
  X1 = X1, X2 = X2, Z = Z, h = 0.08, measures = "tilde_T0_CvM")

result <- measures_nonsimplifyingness_NP(
  X1 = X1, X2 = X2, Z = Z, h = 0.08, measures = "all")

```

se	<i>Standard Error of Model Coefficients</i>
----	---

Description

Generic function to calculate the standard error of estimated coefficients from a model object.

Usage

```
se(object, ...)
```

Arguments

object	A model object.
...	Additional arguments passed to methods.

se.estimated_CKT_kernel	<i>Methods for class 'estimated_CKT_kernel'</i>
-------------------------	---

Description

Methods for class 'estimated_CKT_kernel'

Usage

```

## S3 method for class 'estimated_CKT_kernel'
se(object, progressBar = TRUE, ...)

## S3 method for class 'estimated_CKT_kernel'
confint(object, parm = NULL, level = 0.95, progressBar = TRUE, ...)

## S3 method for class 'estimated_CKT_kernel'
plot(
  x,

```

```

    confint = NULL,
    level = NULL,
    xlim = NULL,
    ylim = c(-1.2, 1.2),
    progressBar = TRUE,
    color_CKT = "black",
    color_confint = "red",
    ...
)

```

Arguments

object, x	an S3 object of class <code>estimated_CKT_kernel</code> .
progressBar	TRUE if a progress bar is plotted if computations of standard errors is needed. Note that in some case, the standard error is already available in the object itself, then no progress bar is needed.
...	other arguments, currently passed to <code>plot.default</code> only for the <code>plot</code> method. These are ignored for the other methods.
parm	ignored for the moment, kept for compatibility with the generic <code>confint</code> method.
level	the confidence level for the confidence intervals. By default, 95% confidence intervals are computed, i.e. <code>level = 0.95</code> .
confint	in case of the <code>plot</code> method, should confidence bands also be plotted?
xlim, ylim	the x,y limits of the plot.
color_CKT, color_confint	the colors respectively for the CKT curve and for the confidence intervals.

Value

`plot` is only called for its side effect and does not return anything.

`se` returns a vector of the same length as the number of points in the input `newZ` that was given to the function [CKT.kernel](#).

`confint` returns a matrix with 2 columns and the same number of rows as the number of points in the input `newZ` that was given to the function [CKT.kernel](#).

See Also

[CKT.kernel](#) which generates objects of class `estimated_CKT_kernel`.

Examples

```

# We simulate from a conditional copula
set.seed(1)
N = 100
# This is a small example for performance reasons.
# For a better example, use:
# N = 800
Z = rnorm(n = N, mean = 5, sd = 2)

```

```

conditionalTau = -0.9 + 1.8 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N , family = 1,
  par = VineCopula::BiCopTau2Par(1 , conditionalTau ))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

newZ = seq(2, 10, by = 1)
estimatedCKT_kernel <- CKT.kernel(
  X1 = X1, X2 = X2, Z = Z,
  newZ = newZ, h = 0.2, kernel.name = "Epa", se = TRUE)

se(estimatedCKT_kernel)
confint(estimatedCKT_kernel, level = 0.9)

plot(estimatedCKT_kernel, confint = TRUE)

```

simpA.kendallReg	<i>Test of the simplifying assumption using the constancy of conditional Kendall's tau</i>
------------------	--

Description

This function computes Kendall's regression, a regression-like model for conditional Kendall's tau. More precisely, it fits the model

$$\Lambda(\tau_{X_1, X_2|Z=z}) = \sum_{j=1}^{p'} \beta_j \psi_j(z),$$

where $\tau_{X_1, X_2|Z=z}$ is the conditional Kendall's tau between X_1 and X_2 conditionally to $Z = z$, Λ is a function from $] - 1, 1]$ to R , $(\beta_1, \dots, \beta_p)$ are unknown coefficients to be estimated and $\psi_1, \dots, \psi_{p'}$ are a dictionary of functions. Then, this function tests the assumption

$$\beta_2 = \beta_3 = \dots = \beta_{p'} = 0,$$

where the coefficient corresponding to the intercept is removed.

Usage

```

simpA.kendallReg(
  X1,
  X2,
  Z,
  vectorZToEstimate = NULL,
  listPhi = list(z = function(z) {
    return(z)
  }),
  typeEstCKT = 4,

```

```

    h_kernel,
    Lambda = function(x) {
        return(x)
    },
    Lambda_deriv = function(x) {
        return(1)
    },
    Lambda_inv = function(x) {
        return(x)
    },
    lambda = NULL,
    h_lambda = h_kernel,
    Kfolds_lambda = 5,
    l_norm = 1
)

## S3 method for class 'simpA_kendallReg_test'
se(object, type, ...)

## S3 method for class 'simpA_kendallReg_test'
coef(object, ...)

## S3 method for class 'simpA_kendallReg_test'
vcov(object, ...)

## S3 method for class 'simpA_kendallReg_test'
print(x, ...)

## S3 method for class 'simpA_kendallReg_test'
plot(x, ylim = c(-1.5, 1.5), ...)

```

Arguments

X1	vector of observations of the first conditioned variable
X2	vector of observations of the second conditioned variable
Z	vector of observations of the conditioning variable
vectorZToEstimate	vector containing the points Z'_i to be used at which the conditional Kendall's tau should be estimated.
listPhi	the list of transformations ϕ to be used.
typeEstCKT	the type of estimation of the kernel-based estimation of conditional Kendall's tau.
h_kernel	the bandwidth used for the kernel-based estimations.
Lambda	the function to be applied on conditional Kendall's tau. By default, the identity function is used.
Lambda_deriv	the derivative of the function Lambda.

<code>Lambda_inv</code>	the inverse function of <code>Lambda</code> .
<code>lambda</code>	the penalization parameter used for Kendall's regression. By default, cross-validation is used to find the best value of <code>lambda</code> if <code>length(listPhi) > 1</code> . Otherwise <code>lambda = 0</code> is used.
<code>h_lambda</code>	bandwidth used for the smooth cross-validation in order to get a value for <code>lambda</code> .
<code>Kfolds_lambda</code>	the number of subsets used for the cross-validation in order to get a value for <code>lambda</code> .
<code>l_norm</code>	type of norm used for selection of the optimal <code>lambda</code> by cross-validation. <code>l_norm=1</code> corresponds to the sum of absolute values of differences between predicted and estimated conditional Kendall's tau while <code>l_norm=2</code> corresponds to the sum of squares of differences.
<code>object, x</code>	an S3 object of class <code>simpA_kendallReg_test</code> .
<code>type</code>	type of standard error to be returned. Possible values are: <ul style="list-style-type: none"> • "coefficients": standard errors for estimated coefficients • "kernel-based CKT": standard errors for the kernel-based estimated values of conditional Kendall's taus • "regression-based CKT": standard errors for the regression-based estimated values of conditional Kendall's taus
<code>...</code>	other arguments, unused
<code>ylim</code>	graphical parameter, see plot

Value

`simpA.kendallReg` returns an S3 object of class `simpA_kendallReg_test`, containing

- `statWn`: the value of the test statistic.
- `p_val`: the p-value of the test.

`plot.simpA_kendallReg_test` returns (invisibly) a matrix with columns `z`, `est_CKT_NP`, `asympt_se_np`, `est_CKT_NP_q025`, `est_CKT_NP_q975`, `est_CKT_reg`, `asympt_se_reg`, `est_CKT_reg_q025`, `est_CKT_reg_q975`. The first column correspond to the grid of values of `z`. The next 4 columns are the NP (kernel-based) estimator of conditional Kendall's tau, with its standard error, and lower/upper confidence bands. The last 4 columns are the equivalents for the estimator based on Kendall's regression.

`plot.simpA_kendallReg_test` plots the kernel-based estimator and its confidence band (in red), and the estimator based on Kendall's regression and its confidence band (in blue).

Usually the confidence band for Kendall's regression is much tighter than the pure non-parametric counterpart. This is because the parametric model is sparser and the corresponding estimator converges faster (even without penalization).

`print.simpA_kendallReg_test` has no return values and is only called for its side effects.

Function `coef.simpA_kendallReg_test` returns the matrix of coefficients with standard errors, `z` values and p-values.

Function `vcov.simpA_kendallReg_test` returns the (estimated) variance-covariance matrix of the estimated coefficients.

Function `se.simpA_kendallReg_test` returns the standard errors of the estimated coefficients or of the estimated conditional Kendall's taus.

References

Derumigny, A., & Fermanian, J. D. (2020). On Kendall's regression. *Journal of Multivariate Analysis*, 178, 104610. (page 7) [doi:10.1016/j.jmva.2020.104610](https://doi.org/10.1016/j.jmva.2020.104610)

See Also

The function to fit Kendall's regression: [CKT.kendallReg.fit](#).

Other tests of the simplifying assumption:

- [simpA.NP](#) in a nonparametric setting
- [simpA.param](#) in a (semi)parametric setting, where the conditional copula belongs to a parametric family, but the conditional margins are estimated arbitrarily through kernel smoothing
- the counterparts of these tests in the discrete conditioning setting: [bCond.simpA.CKT](#) (test based on conditional Kendall's tau) [bCond.simpA.param](#) (test assuming a parametric form for the conditional copula)

Examples

```
# We simulate from a non-simplified conditional copula
set.seed(1)
N = 300
Z = runif(n = N, min = 0, max = 1)
conditionalTau = -0.9 + 1.8 * Z
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

result = simpA.kendallReg(
  X1, X2, Z, h_kernel = 0.03,
  listPhi = list(z = function(z){return(z)}))
print(result)
plot(result)
# Obtain matrix of coefficients, std err, z values and p values
coef(result)
# Obtain variance-covariance matrix of the coefficients
vcov(result)
# Obtain standard errors of the kernel-based estimates of CKT
se(result, type = "kernel-based CKT")
# Obtain standard errors of the regression-based estimates of CKT
se(result, type = "regression-based CKT")

result_morePhi = simpA.kendallReg(
  X1, X2, Z, h_kernel = 0.03,
  listPhi = list(
    z = function(z){return(z)},
    cos10z = function(z){return(cos(10 * z))},
    sin10z = function(z){return(sin(10 * z))},
    `1(z <= 0.4)` = function(z){return(as.numeric(z <= 0.4))},
    `1(z <= 0.6)` = function(z){return(as.numeric(z <= 0.6))})) )
```

```

print(result_morePhi)
plot(result_morePhi)

# We simulate from a simplified conditional copula
set.seed(1)
N = 300
Z = runif(n = N, min = 0, max = 1)
conditionalTau = -0.3
simCopula = VineCopula::BiCopSim(N=N , family = 1,
  par = VineCopula::BiCopTau2Par(1 , conditionalTau ))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

result = simpA.kendallReg(
  X1, X2, Z, h_kernel = 0.03,
  listPhi = list(
    z = function(z){return(z)},
    cos10z = function(z){return(cos(10 * z))},
    sin10z = function(z){return(sin(10 * z))},
    `1(z <= 0.4)` = function(z){return(as.numeric(z <= 0.4))},
    `1(z <= 0.6)` = function(z){return(as.numeric(z <= 0.6))} )
print(result)
plot(result)

```

simpA.NP

Nonparametric testing of the simplifying assumption

Description

This function tests the “simplifying assumption” that a conditional copula

$$C_{1,2|3}(u_1, u_2 | X_3 = x_3)$$

does not depend on the value of the conditioning variable x_3 in a nonparametric setting, where the conditional copula is estimated by kernel smoothing.

Usage

```

simpA.NP(
  X1,
  X2,
  X3,
  testStat,
  typeBoot = "bootNP",
  h,
  nBootstrap = 100,
  kernel.name = "Epanechnikov",

```

```

truncVal = NULL,
numericalInt = list(kind = "legendre", nGrid = 10)
)

```

Arguments

X1	vector of n observations of the first conditioned variable
X2	vector of n observations of the second conditioned variable
X3	vector of n observations of the conditioning variable
testStat	name of the test statistic to be used. Possible values are <ul style="list-style-type: none"> • T1_CvM_Cs3: Equation (3) of (Derumigny & Fermanian, 2017) with the simplified copula estimated by Equation (6) and the weight $w(u_1, u_2, u_3) = \hat{F}_1(u_1)\hat{F}_2(u_2)\hat{F}_3(u_3)$. • T1_CvM_Cs4: Equation (3) of (Derumigny & Fermanian, 2017) with the simplified copula estimated by Equation (7) and the weight $w(u_1, u_2, u_3) = \hat{F}_1(u_1)\hat{F}_2(u_2)\hat{F}_3(u_3)$. • T1_KS_Cs3: Equation (4) of (Derumigny & Fermanian, 2017) with the simplified copula estimated by Equation (6). • T1_KS_Cs4: Equation (4) of (Derumigny & Fermanian, 2017) with the simplified copula estimated by Equation (7). • tilde_T0_CvM: Equation (10) of (Derumigny & Fermanian, 2017). • tilde_T0_KS: Equation (9) of (Derumigny & Fermanian, 2017). • I_chi: Equation (13) of (Derumigny & Fermanian, 2017). • I_2n: Equation (15) of (Derumigny & Fermanian, 2017).
typeBoot	the type of bootstrap to be used (see Derumigny and Fermanian, 2017, p.165). Possible values are <ul style="list-style-type: none"> • boot.NP: usual (Efron's) non-parametric bootstrap • boot.pseudoInd: pseudo-independent bootstrap • boot.pseudoInd.sameX3: pseudo-independent bootstrap without resampling on X_3 • boot.pseudoNP: pseudo-non-parametric bootstrap • boot.cond: conditional bootstrap
h	the bandwidth used for kernel smoothing
nBootstrap	number of bootstrap replications
kernel.name	the name of the kernel
truncVal	the value of truncation for the integral, i.e. the integrals are computed from truncVal to 1-truncVal instead of from 0 to 1. Note that truncVal must be in the interval $[0, 0.5)$, i.e. 0 is allowed but not 0.5. The default is truncVal = NULL, which actually means that truncVal = h if $h < 0.5$ and truncVal = 0 else.
numericalInt	parameters to be given to statmod: : gauss.quad , including the number of quadrature points and the type of interpolation.

Value

a list containing

- `true_stat`: the value of the test statistic computed on the whole sample
- `vect_statB`: a vector of length `nBootstrap` containing the bootstrapped test statistics.
- `p_val`: the p-value of the test.

References

Derumigny, A., & Fermanian, J. D. (2017). About tests of the “simplifying” assumption for conditional copulas. *Dependence Modeling*, 5(1), 154-197. doi:[10.1515/demo20170011](https://doi.org/10.1515/demo20170011)

See Also

Other tests of the simplifying assumption:

- [simpA.param](#) in a (semi)parametric setting, where the conditional copula belongs to a parametric family, but the conditional margins are estimated arbitrarily through kernel smoothing
- [simpA.kendallReg](#): test based on the constancy of conditional Kendall’s tau
- the counterparts of these tests in the discrete conditioning setting: [bCond.simpA.CKT](#) (test based on conditional Kendall’s tau) [bCond.simpA.param](#) (test assuming a parametric form for the conditional copula)

The non-parametric measures of non-simplifyingness [measures_nonsimplifyingness_NP](#).

Examples

```
# We simulate from a conditional copula
set.seed(1)
N = 500
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = -0.9 + 1.8 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1], mean = Z)
X2 = qnorm(simCopula[,2], mean = - Z)

result <- simpA.NP(
  X1 = X1, X2 = X2, X3 = Z,
  testStat = "I_chi", typeBoot = "boot.pseudoInd",
  h = 0.03, kernel.name = "Epanechnikov", nBootstrap = 10)

# In practice, it is recommended to use at least nBootstrap = 100
# with nBootstrap = 200 being a good choice.

print(result$p_val)

set.seed(1)
N = 500
Z = rnorm(n = N, mean = 5, sd = 2)
```

```

conditionalTau = 0.8
simCopula = VineCopula::BiCopSim(N=N , family = 1,
  par = VineCopula::BiCopTau2Par(1 , conditionalTau ))
X1 = qnorm(simCopula[,1], mean = Z)
X2 = qnorm(simCopula[,2], mean = - Z)

result <- simpA.NP(
  X1 = X1, X2 = X2, X3 = Z,
  testStat = "I_chi", typeBoot = "boot.pseudoInd",
  h = 0.08, kernel.name = "Epanechnikov", nBootstrap = 10)
print(result$p_val)

```

simpA.param

Semiparametric testing of the simplifying assumption

Description

This function tests the “simplifying assumption” that a conditional copula

$$C_{1,2|3}(u_1, u_2 | X_3 = x_3)$$

does not depend on the value of the conditioning variable x_3 in a semiparametric setting, where the conditional copula is of the form

$$C_{1,2|3}(u_1, u_2 | X_3 = x_3) = C_{\theta(x_3)}(u_1, u_2),$$

for all $0 \leq u_1, u_2 \leq 1$ and all x_3 . Here, (C_θ) is a known family of copula and $\theta(x_3)$ is an unknown conditional dependence parameter. In this setting, the simplifying assumption can be rewritten as “ $\theta(x_3)$ **does not depend on x_3 , i.e. is a constant function of x_3** ”.

Usage

```

simpA.param(
  X1,
  X2,
  X3,
  family,
  testStat = "T2c",
  typeBoot = "boot.NP",
  h,
  nBootstrap = 100,
  kernel.name = "Epanechnikov",
  truncVal = NULL,
  numericalInt = list(kind = "legendre", nGrid = 10)
)

```

Arguments

X1	vector of n observations of the first conditioned variable
X2	vector of n observations of the second conditioned variable
X3	vector of n observations of the conditioning variable
family	the chosen family of copulas (see the documentation of the class <code>VineCopula::BiCop()</code> for the available families).
testStat	name of the test statistic to be used. The only choice implemented yet is 'T2c'.
typeBoot	the type of bootstrap to be used. (see Derumigny and Fermanian, 2017, p.165). Possible values are <ul style="list-style-type: none"> • "boot.NP": usual (Efron's) non-parametric bootstrap • "boot.pseudoInd": pseudo-independent bootstrap • "boot.pseudoInd.sameX3": pseudo-independent bootstrap without resampling on X_3 • "boot.pseudoNP": pseudo-non-parametric bootstrap • "boot.cond": conditional bootstrap • "boot.paramInd": parametric independent bootstrap • "boot.paramCond": parametric conditional bootstrap
h	the bandwidth used for kernel smoothing
nBootstrap	number of bootstrap replications
kernel.name	the name of the kernel
truncVal	the value of truncation for the integral, i.e. the integrals are computed from truncVal to 1-truncVal instead of from 0 to 1. Note that truncVal must be in the interval $[0, 0.5)$, i.e. 0 is allowed but not 0.5. The default is truncVal = NULL, which actually means that truncVal = h if $h < 0.5$ and truncVal = 0 else.
numericalInt	parameters to be given to statmod: <code>:gauss.quad</code> , including the number of quadrature points and the type of interpolation.

Value

a list containing

- true_stat: the value of the test statistic computed on the whole sample
- vect_statB: a vector of length nBootstrap containing the bootstrapped test statistics.
- p_val: the p-value of the test.

References

Derumigny, A., & Fermanian, J. D. (2017). About tests of the “simplifying” assumption for conditional copulas. *Dependence Modeling*, 5(1), 154-197. doi:10.1515/demo20170011

See Also

Other tests of the simplifying assumption:

- `simpA.NP` in a nonparametric setting
- `simpA.kendallReg`: test based on the constancy of conditional Kendall's tau
- the counterparts of these tests in the discrete conditioning setting: `bCond.simpA.CKT` (test based on conditional Kendall's tau) `bCond.simpA.param` (test assuming a parametric form for the conditional copula)

Examples

```
# We simulate from a conditional copula
set.seed(1)
N = 500
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = -0.9 + 1.8 * pnorm(Z, mean = 5, sd = 2)
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1], mean = Z)
X2 = qnorm(simCopula[,2], mean = - Z)

result <- simpA.param(
  X1 = X1, X2 = X2, X3 = Z, family = 1,
  h = 0.03, kernel.name = "Epanechnikov", nBootstrap = 5)
print(result$p_val)
# In practice, it is recommended to use at least nBootstrap = 100
# with nBootstrap = 200 being a good choice.

set.seed(1)
N = 500
Z = rnorm(n = N, mean = 5, sd = 2)
conditionalTau = 0.8
simCopula = VineCopula::BiCopSim(N=N, family = 1,
  par = VineCopula::BiCopTau2Par(1, conditionalTau))
X1 = qnorm(simCopula[,1], mean = Z)
X2 = qnorm(simCopula[,2], mean = - Z)

result <- simpA.param(
  X1 = X1, X2 = X2, X3 = Z, family = 1,
  h = 0.08, kernel.name = "Epanechnikov", nBootstrap = 5)
print(result$p_val)
```


Index

bCond.estParamCopula, 3, 5, 9
bCond.pobs, 4, 4
bCond.simpA.CKT, 4, 5, 9, 11, 58, 61, 64
bCond.simpA.param, 4, 7, 8, 58, 61, 64
bCond.treeCKT, 5–7, 10, 13, 42, 43
BiCop, 63

CKT.estimate, 11, 12, 16, 18, 22, 27, 33, 36
CKT.fit.GLM, 13, 15, 18, 22, 27, 36, 43, 44
CKT.fit.nNets, 13, 16, 17, 22, 27, 36, 43, 44
CKT.fit.randomForest, 13, 16, 18, 18, 22,
27, 36, 43, 44
CKT.fit.tree, 13, 16, 18, 21, 27, 36, 43, 44
CKT.hCV.Kfolds, 33
CKT.hCV.Kfolds (CKT.hCV.l1out), 22
CKT.hCV.l1out, 22, 33
CKT.kendallReg.fit, 13, 16, 18, 22, 25, 30,
36, 58
CKT.KendallReg.LambdaCV, 26, 28
CKT.kendallReg.predict
(CKT.kendallReg.fit), 25
CKT.kernel, 13, 16, 18, 22, 24, 26, 27, 30, 36,
39, 54
CKT.predict.GLM (CKT.fit.GLM), 15
CKT.predict.kNN, 13, 16, 18, 22, 27, 34, 43,
44
CKT.predict.nNets, 37
CKT.predict.randomForest
(CKT.fit.randomForest), 18
CKT.predict.tree (CKT.fit.tree), 21
CKTmatrix.kernel, 33, 37
coef.simpA_kendallReg_test
(simpA.kendallReg), 55
computeKernelMatrix, 40, 45–47
computeMatrixSignPairs, 24, 29, 41
confint.estimated_CKT_kernel
(se.estimated_CKT_kernel), 53
conv_treeCKT, 42

datasetPairs, 15, 17, 19, 21, 35, 43

estimateCondCDF_matrix, 40, 45
estimateCondCDF_vec, 40, 46
estimateCondQuantiles, 47
estimateNPCondCopula, 13, 48, 50
estimateParCondCopula, 13, 49, 49
estimateParCondCopula_ZIJ
(estimateParCondCopula), 49

gauss.quad, 52, 60, 63
glmnet, 27

matrixInd2matrixCKT, 11
matrixInd2matrixCKT (conv_treeCKT), 42
measures_nonsimplifyingness_NP, 51, 61

ordinalNet, 15

plot, 57
plot.estimated_CKT_kernel, 33
plot.estimated_CKT_kernel
(se.estimated_CKT_kernel), 53
plot.simpA_kendallReg_test
(simpA.kendallReg), 55
print.simpA_kendallReg_test
(simpA.kendallReg), 55

quantile, 6, 11

se, 53
se.estimated_CKT_kernel, 53
se.simpA_kendallReg_test
(simpA.kendallReg), 55
simpA.kendallReg, 7, 9, 27, 55, 61, 64
simpA.NP, 7, 9, 49, 52, 58, 59, 64
simpA.param, 7, 9, 50, 58, 61, 62

tree.control, 19, 21
treeCKT2matrixCKT (conv_treeCKT), 42
treeCKT2matrixInd, 6, 11
treeCKT2matrixInd (conv_treeCKT), 42

`vcov.simpA_kendallReg_test`
 (`simpA.kendallReg`), [55](#)
`VineCopula`, [50](#)
`VineCopula::BiCop`, [50](#)