Package ‘CondCopulas’

March 30, 2022

Type       Package
Title      Estimation and Inference for Conditional Copula Models
Version    0.1.1
License    GPL-3
Encoding   UTF-8
RoxygenNote 7.1.2
Suggests  MASS, knitr, rmarkdown, ggplot2, mvtnorm
Imports    VineCopula, pbapply, glmnet, ordinalNet, tree, nnet, data.tree, statmod, pcaPP
VignetteBuilder knitr
BugReports  https://github.com/AlexisDerumigny/CondCopulas/issues
URL        https://github.com/AlexisDerumigny/CondCopulas
NeedsCompilation no
Author      Alexis Derumigny [aut, cre] (<https://orcid.org/0000-0002-6163-8097>), Jean-David Fermanian [ctb, ths] (<https://orcid.org/0000-0001-5960-5555>), Aleksey Min [ctb] (<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  2022-03-30 17:50:09 UTC
R topics documented:

bCond.estParamCopula .................................................. 2
bCond.pobs ............................................................ 4
bCond.simpA.CKT ....................................................... 5
bCond.simpA.param ...................................................... 7
bCond.treeCKT .......................................................... 9
CKT.estimate ............................................................ 11
CKT.fit.GLM ........................................................... 14
CKT.fit.nNets .......................................................... 16
CKT.fit.randomForest .................................................. 17
CKT.hCV.11out .......................................................... 20
CKT.kendallReg.fit ....................................................... 24
CKT.KendallReg.LambdaCV ............................................. 26
CKT.kernel .............................................................. 29
CKT.predict.kNN ........................................................ 31
CKT.predict.nNets ....................................................... 33
CKTmatrix.kernel ....................................................... 34
computeKernelMatrix ................................................... 36
computeMatrixSignPairs ................................................. 37
conv_treeCKT ........................................................... 38
datasetPairs ............................................................ 40
estimateCondCDF_matrix ................................................. 41
estimateCondCDF_vec ................................................... 42
estimateCondQuantiles ................................................ 43
estimateNPCondCopula .................................................. 44
estimatePartCondCopula ................................................ 46
simpA.kendallReg ......................................................... 48
simpA.NP ............................................................... 50
simpA.param ........................................................... 53

Index ................................................................. 56

bCond.estParamCopula  Estimation of the conditional parameters of a parametric conditional copula with discrete conditioning events.

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 \( A_1, ..., A_p \) of the probability space, and that for each \( k = 1,...,p \), the conditional copula is parametric according to the following model

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

**Usage**

\[
\text{bCond.estParamCopula(U1, U2, family, partition)}
\]

**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 \( n \times p \), where \( p \) is the number of conditioning events that are considered. \( \text{partition}[i,j] \) 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**


**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**

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

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

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**


**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 = NA, 
    matrixInd = NA, 
    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*p of observations of the conditioned variables.

XJ  
    matrix of size n*(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

- 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.

methodPvalue: method for computing the p-value

- covMatrix: by computation of the covariance matrix of the random vector \((\tau_{i,k}|X_j \in A_j, 1 \leq i, k \leq p, 1 \leq j \leq m)\).
- bootNP: by the usual non-parametric bootstrap
- bootInd: by the independent bootstrap

nBootstrap: number of bootstrap replications (Only used if `methodPvalue` is not `covMatrix`).

Value

a list with the following components

- p.value the estimated p-value.
- stat the test statistic.
- treeCKT the estimated tree if `matrixInd` is not provided.
- vec_statB the vector of bootstrapped statistics if `methodPvalue` is not `covMatrix`.

Author(s)

Alexis Derumigny, Jean-David Fermanian and Aleksey Min
References

See Also
bCond.simpA.param for a test of this simplifying assumption in a parametric framework.
bCond.treeCKT provides the binary tree that is used in this function (if matrixInd is not provided).

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

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

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. partition[i,j] 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
• \( T_{2c\_par} \sum_{box} (\theta_0 - \theta(box))^2 \)
• \( T_{2c\_tau} \) 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


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

```r
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$p_val)
```
```r
n = 800
Z = stats::runif(n = n)
CKT = 0.1
family = 3
simCopula = VineCopula::BiCopSim(N = n,
   par = VineCopula::BiCopTau2Par(CKT, family = family), family = family)
X1 = simCopula[,1]
X2 = simCopula[,2]
partition = cbind(Z <= 0.3, Z > 0.3 & Z <= 0.5, Z > 0.5)
result = bCond.simpA.param(X1 = X1, X2 = X2,
   partition = partition, family = family, typeBoot = "boot.NP")
print(result$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 \( X_I \) (the conditioned variables) and the second one contains the observations of \( X_J \) (the conditioning variables). The goal of this procedure is to find which of the variables in \( X_J \) have important influence on the dependence between the components of \( X_I \), (measured by the Kendall’s tau).

**Usage**

```r
bCond.treeCKT(
   XI = ,
   XJ = ,
   minCut = 0,
   minProb = 0.01,
   minSize = minProb * length(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 $P_{\text{dim}(XJ)}$), and the conditional Kendall’s tau conditionally to it.

Value

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

References


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

```r
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 = 50, verbose = 2)

# 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

Estimation of conditional Kendall’s tau between two variables $X_1$ and $X_2$ given $Z = z$

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(
  observedX1, observedX2, observedZ,
  newZ = observedZ, methodEstimation, h,
  listPhi = if(methodEstimation == "kendallReg")
    {list( function(x){return(x)} ,
       function(x){return(x^2)} ,
       function(x){return(x^3)} )
     } else {list(identity)} ,
  ...
)

Arguments

- **observedX1**: a vector of $n$ observations of the first variable
- **observedX2**: a vector of $n$ observations of the second variable
- **observedZ**: 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.
  - If observedZ is a vector, then newZ must be a vector as well.
  - If observedZ is a matrix, then newZ must be a matrix as well, with the same number of columns (= the dimension of $Z$).
methodEstimation

method for estimating the conditional Kendall’s tau. Possible estimation methods are:

- "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.

Value

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

References


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)
CKT.estimate

```r
simCopula = VineCopula::BiCopSim(N=N , family = 1, 
    par = VineCopula::BiCopTau2Par(1 , conditionalTau ))
X1 = qnorm(simCopula[,1])
X2 = qnorm(simCopula[,2])

ewZ = seq(2,10,by = 0.1)
h = 0.1
estimatedCKT_tree <- CKT.estimate(
    observedX1 = X1, observedX2 = X2, observedZ = Z,
    newZ = newZ,
    methodEstimation = "tree", h = h)

estimatedCKT_rf <- CKT.estimate(
    observedX1 = X1, observedX2 = X2, observedZ = Z,
    newZ = newZ,
    methodEstimation = "randomForest", h = h)

estimatedCKT_GLM <- CKT.estimate(
    observedX1 = X1, observedX2 = X2, observedZ = 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(
    observedX1 = X1, observedX2 = X2, observedZ = Z,
    newZ = newZ,
    methodEstimation = "nearestNeighbors", h = h,
    number_nn = c(50,80, 100, 120,200),
    partition = 4)

estimatedCKT_nNet <- CKT.estimate(
    observedX1 = X1, observedX2 = X2, observedZ = Z,
    newZ = newZ,
    methodEstimation = "neuralNetwork", h = h,
)

estimatedCKT_kernel <- CKT.estimate(
    observedX1 = X1, observedX2 = X2, observedZ = Z,
    newZ = newZ,
    methodEstimation = "kernel", h = h,
)

estimatedCKT_kendallReg <- CKT.estimate(
    observedX1 = X1, observedX2 = X2, observedZ = 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",
```

```
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

\[
\tau_{1,2|Z} = 2 \Lambda (\beta_0 + \beta_1 \phi_1(Z) + ... + \beta_p \phi_p(Z))
\]

for a link function \( \Lambda \), and \( p \) real-valued functions \( \phi_1, ..., \phi_p \). The function **CKT.predict.GLM** predicts the values of conditional Kendall’s tau for some values of the conditioning variable \( Z \).

Usage

```r
CKT.fit.GLM(
  datasetPairs,
  designMatrix = datasetPairs[, 2:(ncol(datasetPairs) - 3), drop = FALSE],
  link = "logit",
  ...
)
CKT.predict.GLM(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 model. It should have the same number of rows as the `datasetPairs`.
- **link** link function, can be one of logit, probit, cloglog, cauchit).
- **...** other parameters passed to `ordinalNet::ordinalNet()`.
- **fit** result of a call to `CKT.fit.GLM`
- **newZ** new matrix of observations of the conditioning vector \( Z \), with the same number of variables and same names as the `designMatrix` 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**


**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**

```r
# 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

Estimation of conditional Kendall’s taus by model averaging of neural networks

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

```r
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
  • 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


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  
Fit a Random Forest that can be used for the estimation of conditional Kendall's tau.
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

```r
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 `tree::tree.control()` for more details.
- `mincut` the minimum number of observations (of pairs) in a node See `tree::tree.control()` for more details.
- `nObs_per_Tree` number of observations kept in each tree.
CKT.fit.randomForest

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


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")
CKT.fit.tree

Estimation of conditional Kendall’s taus using a classification tree

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 tree::tree.control() for more details.

mincut the minimum number of observations (of pairs) in a node See tree::tree.control() for more details.

fit result of a call to CKT.fit.tree

newZ new matrix of observations, with the same number of variables and same names as the designMatrix 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


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...
These functions aim 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

```r
CKT.hCV.l1out(
  observedX1, observedX2, observedZ, range_h, matrixSignsPairs, nPairs = 10 * length(observedX1), typeEstCKT = 4, kernel.name = "Epa", progressBar = TRUE, verbose = FALSE )
```

```r
CKT.hCV.Kfolds(
  range_h, matrixSignsPairs, observedZ, ZToEstimate, typeEstCKT = 4, kernel.name = "Epa", Kfolds = 5, progressBar = TRUE, verbose = FALSE )
```

### Arguments

- `observedX1` a vector of $n$ observations of the first variable
- `observedX2` a vector of $n$ observations of the second variable
- `observedZ` observed $Z$ vector of observed values of $Z$. If $Z$ is multivariate, then this is a matrix whose rows correspond to the observations of $Z$.
- `range_h` vector containing possible values for the bandwidth.
- `matrixSignsPairs` square matrix of signs of all pairs, produced by `computeMatrixSignPairs`.
- `nPairs` number of pairs used in the cross-validation criteria.
- `typeEstCKT` type of estimation of the conditional Kendall’s tau.
- `kernel.name` name of the kernel used for smoothing. Possible choices are “Gaussian” (Gaussian kernel) and “Epa” (Epanechnikov kernel).
progressBar if TRUE, a progressbar for each h is displayed to show the progress of the computation.
verbose if TRUE, print the score of each h during the procedure.
ZToEstimate 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.
Kfolds 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


See Also

CKT.kernel for the corresponding estimator of conditional Kendall’s tau by kernel smoothing.

Examples

```r
# 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(2,10,by = 0.1)
range_h = 3:10
matrixSignsPairs = computeMatrixSignPairs(vectorX1 = X1, vectorX2 = X2)
resultCV <- CKT.hCV.Kfolds(range_h = range_h,
                   matrixSignsPairs = matrixSignsPairs, observedZ = Z, ZToEstimate = newZ)
plot(range_h, resultCV$scores, type = "b")
```
Fit Kendall's regression, a GLM-type model for conditional Kendall's tau

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 $Z = z$, $\Lambda$ is a function from $[-1, 1]$ to $\mathbb{R}$, $(\beta_1, \ldots, \beta_p)$ are unknown coefficients to be estimated and $\psi_1, \ldots, \psi_{p'}$ are a dictionary of functions. To estimate beta, we used the penalized estimator which is defined as the minimizer of the following criteria

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

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

```r
CKT.kendallReg.fit(
  observedX1, observedX2, observedZ, 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, ...
)
CKT.kendallReg.predict(fit, newZ, lambda = NULL, Lambda_inv = identity)
```
**Arguments**

- **observedX1**: a vector of $n$ observations of the first variable $X_1$.
- **observedX2**: a vector of $n$ observations of the second variable $X_2$.
- **observedZ**: 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 intermediary dataset of observations of $Z$ at which the conditional Kendall’s tau should be estimated.
- **designMatrixZ**: the transformation of the $Z$ToEstimate that will be used as predictors. By default, no transformation is applied.
- **newZ**: the new observations of the conditioning variable.
- **h_kernel**: bandwidth used for the first step of kernel smoothing.
- **Lambda**: the function to be applied on conditional Kendall’s tau. By default, the identity function is used.
- **Lambda_inv**: the functional inverse of $\Lambda$. By default, the identity function is used.
- **lambda**: the regularization parameter. If NULL, then it is chosen by K-fold cross validation. Internally, cross-validation is performed by the function `CKT.kendallReg.LambdaCV`.
- **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 $\lambda$ by cross-validation. $l_{\text{norm}}=1$ corresponds to the sum of absolute values of differences between predicted and estimated conditional Kendall’s tau while $l_{\text{norm}}=2$ corresponds to the sum of squares of differences.
- **h_lambda**: the smoothing bandwidth used in the cross-validation procedure to choose $\lambda$.
- **other arguments to be passed to `CKT.kernel` for the first step (kernel-based) estimator of conditional Kendall’s tau.**
- **fit**: the fitted model, obtained by a call to `CKT.kendallReg.fit`.

**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**

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(
    observedX1 = X1, observedX2 = X2, observedZ = 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")
where \( \tau_{X_1, X_2|Z=z} \) is the conditional Kendall’s tau between \( X_1 \) and \( X_2 \) conditionally to \( Z = z \), \( \Lambda \) is a function from \( [-1, 1] \) to \( \mathbb{R} \), \( (\beta_1, \ldots, \beta_p) \) are unknown coefficients to be estimated and \( \psi_1, \ldots, \psi_p \) are a dictionary of functions. To estimate \( \beta \), we used the penalized estimator which is defined as the minimizer of the following criteria

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

This function chooses the penalization parameter \( \lambda \) by cross-validation.

Usage

```r
CKT.KendallReg.LambdaCV(
  observedX1, observedX2, observedZ, 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"
)
```

Arguments

- **observedX1** a vector of \( n \) observations of the first variable \( X_1 \).
- **observedX2** a vector of \( n \) observations of the second variable \( X_2 \).
- **observedZ** 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 \( Z \) 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

- "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.

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


See Also

The main fitting function `CKT.kendallReg.fit`.

Examples

```r
# 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(
  observedX1 = X1, observedX2 = X2, observedZ = Z,
  ZToEstimate = newZ, h_lambda = 2)
```
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

```r
CKT.kernel(
  observedX1, observedX2, observedZ, newZ, h,
  kernel.name = "Epa",
  methodCV = "Kfolds",
  Kfolds = 5,
  nPairs = 10 * length(observedX1),
  typeEstCKT = 4,
  progressBar = TRUE
)
```

Arguments

- `observedX1`: a vector of n observations of the first variable
- `observedX2`: a vector of n observations of the second variable
- `observedZ`: a vector of n observations of the conditioning variable, or a matrix with n rows of observations of the conditioning vector
- `newZ`: the new data of observations of $Z$ at which the conditional Kendall’s tau should be estimated.
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.

name of the kernel used for smoothing. Possible choices are "Gaussian" (Gaussian kernel) and "Epa" (Epanechnikov kernel).

method used for the cross-validation. Possible choices are leave-one-out and Kfolds.

number of subsamples used, if methodCV = "Kfolds".

number of pairs used in the cross-validation criteria, if methodCV = "leave-one-out".

type of estimation of the conditional Kendall's tau. Default is 4. 1 and 3 produced biased estimator while 2 does not attain the full range \([-1, 1]\).

if TRUE, a progressbar for each h is displayed to show the progress of the computation.

a list with two components

- 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.)


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.l1out for manual selection of the bandwidth by leave-one-out or K-folds cross-validation.

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)
estimatedCKT_kernel <- CKT.kernel(
  observedX1 = X1, observedX2 = X2, observedZ = 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")

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,
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.
norm.Lp 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 length(number.nn) > 1. 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

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.GLMM, CKT.kernel, CKT.kendallReg.fit, and the more general wrapper CKT.estimate.

Examples

```r
# 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  

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

Description

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

Usage

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

fit                    result of a call to CKT.fit.nNet
newZ                  new matrix of observations, with the same number of variables and same names as the designMatrix 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.

Usage

CKTmatrix.kernel(  
    dataMatrix,  
    observedZ,  
    gridZ,  
    averaging = "no",  
    blockStructure = NULL,  
    h,  
    kernel.name = "Epa",  
    typeEstCKT = 4  
)
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
  • 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 \((n,n,\text{length(gridZ)})\), 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 \((\text{length(blockStructure)},\text{length(blockStructure)},\text{length(gridZ)})\), 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

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

```r
# 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_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**

**Computing the kernel matrix**

**Description**

This function computes a matrix of dimensions `(length(observedX3), length(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.
computeMatrixSignPairs

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 (length(observedX), length(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 \( n \times n \) 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])

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

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

Value

- The function `treeCKT2matrixInd` returns a matrix of size \( N \times m \) which component \([i,j]\) is

\[
1\{X_{i,J} \in A_{j,J}\}
\]

- The function `matrixInd2matrixCKT` and `treeCKT2matrixCKT` return a matrix of size \(|I| \times (|I|-1) \times m\) 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

```r
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

Construct a dataset of pairs of observations for the estimation of conditional Kendall’s tau

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: \texttt{CKT.fit.tree}, \texttt{CKT.fit.randomForest}, \texttt{CKT.fit.GLM}, \texttt{CKT.fit.nNets}, \texttt{CKT.predict.kNN}.

Usage

\begin{verbatim}
datasetPairs(
  X1, X2, Z, h, cut = 0.9, onlyConsecutivePairs = FALSE, nPairs = NULL)
\end{verbatim}

Arguments

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

Value

A matrix with \((4+\text{dim}Z)\) columns and \(n*(n-1)/2\) rows if \texttt{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 \(1+\text{dim}Z\) contain the mean value of \(Z\) (the conditioning variables);
- column \(2+\text{dim}Z\) contains the value of the kernel \(K_h(z_j - z_i)\);
- column \(3+\text{dim}Z\) and \(4+\text{dim}Z\) contain the corresponding values of \(i\) and \(j\).
estimateCondCDF_matrix

References

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) \( K_h(X3[i] - U3[j]) \) such as given by computeKernelMatrix.
estimateCondCDF_vec

Details

This function is supposed to be used with computeKernelMatrix. Assume that we observe a sample \((X_{i,1}, X_{i,3}), i = 1, \ldots, n\). 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

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

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

Value

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

Examples

```r
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(\text{observedX} = Y[,2], \text{newX} = newY2,
kernel = "Gaussian", h = 0.8)
# In this matrix, there are the estimated conditional cdf at points given by newY1
# conditionally to the points given by newY2.
matrixCondCDF = estimateCondCDF_matrix(\text{observedX1} = Y[,1],
newX1 = newY1, matrixK)
matrixCondCDF
```

estimateCondCDF_vec

Compute kernel-based conditional marginal (univariate) cdfs

Description

This function computes an estimate of the conditional (marginal) cdf of \(X_1\) given a conditioning variable \(X_3\). This function is supposed to be used with computeKernelMatrix. Assume that we observe a sample \((X_{i,1}, X_{i,3}), i = 1, \ldots, n\). 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

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

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 \(K_h(X_{i,3} - x_3)\) such as the one produced by computeKernelMatrix(\(\text{observedX3, newX3, kernel, h}\)).

Usage

```r
estimateCondCDF_vec(\text{observedX1, newX1, matrixK3})
```
estimateCondQuantiles

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) \( \left( K_h(X3[i] - U3[j]) \right)_{i,j} \) 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

```r
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 \((X_{i,1}, X_{i,3}), i = 1, \ldots, n\). 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

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

Arguments

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

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 \(\text{newX3}[j]\) is the vector that was used to construct \(\text{matrixK3}\).

Examples

```r
Y = MASS::mvrnorm(n = 100, mu = c(0,0), Sigma = cbind(c(1, 0.9), c(0.9, 1)))
matrixK = computeKernelMatrix(observationsX = Y[,2], newX = c(0, 1, 2.5),
                              kernel = "Gaussian", h = 0.8)
matrixnp = estimateCondQuantiles(observationsX1 = Y[,2],
                                   probsX1 = c(0.3, 0.5), matrixK3 = matrixK)
matrixnp
```

estimateNPCondCopula \hspace{1em} \textit{Compute a kernel-based estimator of the conditional copula}

Description

Assuming that we observe a sample \((X_{i,1}, X_{i,2}, X_{i,3}), i = 1, \ldots, n\), 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

```r
estimateNPCondCopula(
  observedX1,
  observedX2,
  observedX3,
  U1,       
  U2,       
  newX3,    
  kernel,   
  h
)
```
**estimateNPCondCopula**

**Arguments**

- `observedX1` a vector of observations of size n
- `observedX2` a vector of observations of size n
- `observedX3` a vector 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.

**Value**

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

**References**


**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**

```r
# 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(observedX1 = X1, observedX2 = X2, observedX3 = X3, U1_ = grid, U2_ = grid, newX3 = c(2, 5, 7), kernel = "Gaussian", h = 0.8)
arrayEst
```
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 \( (C_\theta) \), 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(
  observedX1, observedX2, observedX3, newX3, family, method = "mle", h
)

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

Arguments

- `observedX1`: a vector of \( n \) observations of the first conditioned variable
- `observedX2`: a vector of \( n \) observations of the second conditioned variable
- `observedX3`: 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 package `VineCopula`.
- `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
- `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 \( i = 1, \ldots, n \).
- `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 \( i = 1, \ldots, n \).
Value

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

References


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

```r
# 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(
    observedX1 = X1, observedX2 = X2, observedX3 = 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
```
simpA.kendallReg

Test of the simplifying assumption using the constancy of conditional Kendall’s tau

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 \(Z = z\), \(\Lambda\) is a function from \([-1, 1]\) to \(\mathbb{R}\), \((\beta_1, \ldots, \beta_{p'})\) are unknown coefficients to be estimated and \((\psi_1, \ldots, \psi_{p'})\) are a dictionary of functions. Then, this function tests the assumption

\[
\beta_2 = \beta_3 = \ldots = \beta_{p'} = 0,
\]

where the coefficient corresponding to the intercept is removed.

Usage

```r
simpA.kendallReg(
  X1,
  X2,
  Z,
  vectorZToEstimate = NULL,
  listPhi = list(function(x) { return(x) }, function(x) { return(x^2) },
                 function(x) { return(x^3) }),
  typeEstCKT = 4,
  h_kernel,
  Lambda = function(x) { return(x) },
  Lambda_deriv = function(x) { return(1) },
  lambda = NULL,
  h_lambda = h_kernel,
  Kfolds_lambda = 5,
  l_norm = 1
)
```

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.
**simpA.kendallReg**

- `listPhi`: the list of transformations \( \phi \) 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 \( \Lambda \).
- `lambda`: the penalization parameter used for Kendall’s regression. By default, cross-validation is used to find the best value of \( \lambda \).
- `h_lambda`: bandwidth used for the smooth cross-validation in order to get a value for \( \lambda \).
- `Kfolds_lambda`: the number of subsets used for the cross-validation in order to get a value for \( \lambda \).
- `l_norm`: type of norm used for selection of the optimal \( \lambda \) by cross-validation. \( l_{\text{norm}}=1 \) corresponds to the sum of absolute values of differences between predicted and estimated conditional Kendall’s tau while \( l_{\text{norm}}=2 \) corresponds to the sum of squares of differences.

**Value**

A list containing

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

**References**


**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

```r
# We simulate from a 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(
        function(x){return(x)}, function(x){return(x^2)},
        function(x){return(x^3)}, function(x){return(x^4)},
        function(x){return(x^5)},
        function(x){return(cos(x))}, function(x){return(sin(x))},
        function(x){return(as.numeric(x <= 0.4))},
        function(x){return(as.numeric(x <= 0.6))} )
print(result$p_val)
```

```r
# We simulate from a 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(
        function(x){return(x)}, function(x){return(x^2)},
        function(x){return(x^3)}, function(x){return(x^4)},
        function(x){return(x^5)},
        function(x){return(cos(x))}, function(x){return(sin(x))},
        function(x){return(as.numeric(x <= 0.4))},
        function(x){return(as.numeric(x <= 0.6))} )
print(result$p_val)
```
simpA.NP

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 = h,
  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

\textbf{testStat} name of the test statistic to be used. Possible values are

- \textbf{T1\textunderscore CvM\textunderscore Cs3}: Equation (3) of (Derumigny & Fermanian, 2017) with the simplified copula estimated by Equation (6) and the weight \( w(u_1, u_2, u_3) = \hat{F}_1(u_1)\hat{F}_2(u_2)\hat{F}_3(u_3) \).
- \textbf{T1\textunderscore CvM\textunderscore Cs4}: Equation (3) of (Derumigny & Fermanian, 2017) with the simplified copula estimated by Equation (7) and the weight \( w(u_1, u_2, u_3) = \hat{F}_1(u_1)\hat{F}_2(u_2)\hat{F}_3(u_3) \).
- \textbf{T1\textunderscore KS\textunderscore Cs3}: Equation (4) of (Derumigny & Fermanian, 2017) with the simplified copula estimated by Equation (6).
- \textbf{T1\textunderscore KS\textunderscore Cs4}: Equation (4) of (Derumigny & Fermanian, 2017) with the simplified copula estimated by Equation (7).
- \textbf{tilde\textunderscore T0\textunderscore CvM}: Equation (10) of (Derumigny & Fermanian, 2017).
- \textbf{tilde\textunderscore T0\textunderscore KS}: Equation (9) of (Derumigny & Fermanian, 2017).
- \textbf{I\textunderscore chi}: Equation (13) of (Derumigny & Fermanian, 2017).
- \textbf{I\textunderscore 2n}: Equation (15) of (Derumigny & Fermanian, 2017).

\textbf{typeBoot} the type of bootstrap to be used (see Derumigny and Fermanian, 2017, p.165). Possible values are

- \textbf{boot\textunderscore 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.

`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


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)

Examples

```r
# 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, $(C_\theta)$ 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**

```r
simpA.param(
  X1,
  X2,
)```

---

**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, $(C_\theta)$ 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**

```r
simpA.param(
  X1,
  X2,
)```
simpA.param

X3,
family,
testStat = "T2c",
typeBoot = "boot.NP",
h,
nBootstrap = 100,
kernl.name = "Epanechnikov",
truncVal = h,
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 VineCopula::BiCop() 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
  • "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
kernl.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.
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**


**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**

```r
# 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, 2, 4, 8
bCond.pobs, 3, 4
bCond.simpA.CKT, 3, 5, 8, 10, 49, 52, 55
bCond.simpA.param, 3, 7, 7, 49, 52, 55
bCond.treeCKT, 4, 6, 7, 9, 12, 38, 39
BiCop, 54

CKT.estimate, 10, 11, 15, 17, 21, 26, 30, 33
CKT.fit.GLM, 12, 14, 17, 21, 26, 33, 40, 41
CKT.fit.nNets, 12, 15, 16, 21, 26, 33, 40, 41
CKT.fit.randomForest, 12, 15, 17, 17, 21, 26, 33, 40, 41
CKT.fit.tree, 12, 15, 17, 20, 26, 33, 40, 41
CKT.hCV.Kfolds (CKT.hCV.l1out), 21
CKT.hCV.l1out, 21, 30
CKT.kendallReg.fit, 12, 15, 17, 21, 24, 28, 33, 49
CKT.KendallReg.LambdaCV, 25, 26
CKT.kendallReg.predict (CKT.kendallReg.fit), 24
CKT.kernel, 12, 15, 17, 21, 23, 25, 26, 29, 33, 35
CKT.predict.GLM (CKT.fit.GLM), 14
CKT.predict.KNN, 12, 15, 17, 21, 26, 31, 40, 41
CKT.predict.nNets.33
CKT.predict.randomForest (CKT.fit.randomForest), 17
CKT.predict.tree (CKT.fit.tree), 20
CKTmatrix.kernel, 30, 34
computeKernelMatrix, 36, 41–44
computeMatrixSignPairs, 22, 28, 37
conv_treeCKT, 38

datasetPairs, 14, 16, 18, 20, 32, 40

estimateCondCDF_matrix, 37, 41
estimateCondCDF_vec, 37, 42
estimateCondQuantiles, 43
estimateNPCondCopula, 12, 44, 47

estimateParCondCopula, 12, 45, 46
(estimateParCondCopula_ZIJ)
gauss.quad, 52, 54
glmnet, 25

matrixInd2matrixCKT, 10
(matrixInd2matrixCKT (conv_treeCKT), 38

ordinalNet, 14
quantile, 6, 10

simpA.kendallReg, 7, 8, 26, 48, 52, 55
simpA.NP, 7, 8, 45, 49, 50, 55
simpA.param, 7, 8, 47, 49, 52, 53

tree.control, 18, 20
treeCKT2matrixCKT (conv_treeCKT), 38
treeCKT2matrixInd, 6, 10
treeCKT2matrixInd (conv_treeCKT), 38

VineCopula, 46