Package ‘kpcalg’

October 13, 2022

Date 2017-01-19
Title Kernel PC Algorithm for Causal Structure Detection
Description Kernel PC (kPC) algorithm for causal structure learning and causal inference using graphical models. kPC is a version of PC algorithm that uses kernel based independence criteria in order to be able to deal with non-linear relationships and non-Gaussian noise.

Version 1.0.1
Author Petras Verbyla, Nina Ines Bertille Desgranges, Lorenz Wernisch
Maintainer Petras Verbyla <petras.verbyla@mrc-bsu.cam.ac.uk>
Imports pcalg, energy, kernlab, parallel, mgcv, RSpectra, methods, graph, stats, utils
Suggests Rgraphviz, knitr
VignetteBuilder knitr
License GPL (>= 2)
Depends R (>= 3.0.2)
LazyData TRUE
RoxygenNote 5.0.1
NeedsCompilation no
Repository CRAN
Date/Publication 2017-01-22 12:38:35

R topics documented:

- dcv.gamma .................................................. 2
- frml.additive.smooth ..................................... 3
- frml.full.smooth ........................................ 4
- hsic.clust .................................................. 5
- hsic.gamma .................................................. 6
- hsic.perm .................................................. 8
- hsic.test .................................................. 9
- kernelCltest .............................................. 11
- kpc ......................................................... 13
Test to check the independence between two variables \(x\) and \(y\) using the Distance Covariance. The \texttt{dcov.gamma()} function, uses Distance Covariance independence criterion with gamma approximation to test for independence between two random variables.

### Usage

\[
dcov.gamma(x, y, index = 1, numCol = 100)
\]

### Arguments

- **x**: data of first sample
- **y**: data of second sample
- **index**: exponent on Euclidean distance, in (0,2]
- **numCol**: Number of columns used in incomplete Singular Value Decomposition

### Details

Let \(x\) and \(y\) be two samples of length \(n\). Gram matrices \(K\) and \(L\) are defined as:

\[
K_{i,j} = \|x_i - x_j\|^s
\]

and

\[
L_{i,j} = \|y_i - y_j\|^s, \text{ where } 0 < s < 2.
\]

Let \(H_{i,j} = \delta_{i,j} - \frac{1}{n}\). Let \(A = HKH\) and \(B = HLH\), then

\[
nV^2 = \frac{1}{n^2} \sum A_{i,j}B_{i,j}.
\]

For more detail: \texttt{dcov.test} in package energy. Gamma test compares \(nV^2(x,y)\) with the \(\alpha\) quantile of the gamma distribution with mean and variance same as \(nV_n^2\) under independence hypothesis.

### Value

\texttt{dcov.gamma()} returns a list with class htest containing:

- **method**: description of test
- **statistic**: observed value of the test statistic
- **estimate**: \(nV^2(x,y)\)
- **estimates**: a vector: \([nV^2(x,y), \text{mean of } nV^2(x,y), \text{variance of } nV^2(x,y)]\)
- **replicates**: replicates of the test statistic
- **p.value**: approximate p-value of the test
- **data.name**: description of data
frml.additive.smooth

Author(s)

Petras Verbyla (<petras.verbyla@mrc-bsu.cam.ac.uk>) and Nina Ines Bertille Desgranges

References


See Also

hsic.perm, hsic.clust, hsic.gamma, dcov.test, kernelCItest

Examples

library(energy)
set.seed(10)
#independence
x <- runif(300)
y <- runif(300)
hsic.gamma(x,y)
hsic.perm(x,y)
dcov.gamma(x,y)
dcov.test(x,y)

#uncorelated but not dependent
z <- 10*(runif(300)-0.5)
w <- z^2 + 10*runif(300)
cor(z,w)
hsic.gamma(z,w)
hsic.perm(z,w)
dcov.gamma(z,w)
dcov.test(z,w)

---

frml.additive.smooth  Formula for GAM without crossterms

Description

Creates a formula for gam to be used in regrXonS. For data \( X = (X_1, ..., X_n, X_{n+1}, ..., X_m) \), variable to be regressed \( X_i, i=1,...n \) and variables to regress on \( S = X_{n+1}, ..., X_m \) creates formula \( X_i \sim s(X_{n+1}) + ... + s(X_m) \).

Usage

frml.additive.smooth(target.ind, pred.inds, var.str = "x")
### frml.full.smooth

**Arguments**
- `target.ind` integer, number for the variable to be regressed
- `pred.inds` integer(s), number(s) for the variable(s) on which we regress
- `var.str` name of variables used to create formula, default is "x"

**Value**

`formula.additive.smooth()` returns a formula $X_i \sim s(X_{n+1}) + \ldots + s(X_m)$

**Author(s)**

Petras Verbyla (<petras.verbyla@mrc-bsu.cam.ac.uk>)

**See Also**

`regrXonS`

---

**Description**

Creates a formula for `gam` to be used in `regrXonS`. For data $X = (X_1, \ldots, X_n, X_{n+1}, \ldots, X_m)$, variable to be regressed $X_i, i=1\ldots n$ and variables to regress on $S = X_{n+1}, \ldots, X_m$ creates formula $X_i \sim s(X_{n+1}, \ldots, X_m)$.

**Usage**

`frml.full.smooth(target.ind, pred.inds, var.str = "x")`

**Arguments**
- `target.ind` integer, number for the variable to be regressed
- `pred.inds` integer(s), number(s) for the variable(s) on which we regress
- `var.str` name of variables used to create formula, default is "x"

**Value**

`formula.full.smooth()` returns a formula $X_i \sim s(X_{n+1}, \ldots, X_m)$

**Author(s)**

Petras Verbyla (<petras.verbyla@mrc-bsu.cam.ac.uk>)

**See Also**

`regrXonS`
**hsic.clust**  
*HSIC cluster permutation conditional independence test*

**Description**
Conditional independence test using HSIC and permutation with clusters.

**Usage**
```r
hsic.clust(x, y, z, sig = 1, p = 100, numCluster = 10, numCol = 50, 
eps = 0.1, paral = 1)
```

**Arguments**
- `x`: first variable  
- `y`: second variable  
- `z`: set of variables on which we condition  
- `sig`: the width of the Gaussian kernel  
- `p`: the number of permutations  
- `numCluster`: number of clusters for clustering `z`  
- `numCol`: maximum number of columns that we use for the incomplete Cholesky decomposition  
- `eps`: normalization parameter for HSIC cluster test  
- `paral`: number of cores used

**Details**
Let \( x \) and \( y \) be two samples of length \( n \). Gram matrices \( K \) and \( L \) are defined as:
\[
K_{i,j} = \exp\left(\frac{(x_i - x_j)^2}{\sigma^2}\right),
\]
\[
L_{i,j} = \exp\left(\frac{(y_i - y_j)^2}{\sigma^2}\right),
\]
\[
M_{i,j} = \exp\left(\frac{(z_i - z_j)^2}{\sigma^2}\right).
\]
\( H_{i,j} = \delta_{i,j} - \frac{1}{n} \). Let \( A = HKH \), \( B = HLH \) and \( C = HMH \). \( HSIC(X,Y|Z) = \frac{1}{n^2} Tr(AB - 2AC(C+\epsilon I)^{-2}CB + AC(C+\epsilon I)^{-2}CBC(C+\epsilon I)^{-2}) \). Permutation test clusters \( Z \) and then permutes \( Y \) in the clusters of \( Z \) \( p \) times to get \( Y(p) \) and calculates \( HSIC(X,Y(p)|Z) \).

**Value**
- `hsic.clust()` returns a list with class `htest` containing:
  - `method`: description of test  
  - `statistic`: observed value of the test statistic  
  - `estimate`: \( HSIC(x,y) \)  
  - `estimates`: a vector: \([HSIC(x,y),\ \text{mean of } HSIC(x,y),\ \text{variance of } HSIC(x,y)]\)  
  - `replicates`: replicates of the test statistic  
  - `p.value`: approximate p-value of the test  
  - `data.name`: description of data
Author(s)

Petras Verbyla (<petras.verbyla@mrc-bsu.cam.ac.uk>) and Nina Ines Bertille Desgranges

References


See Also

hsic.gamma, hsic.perm, kernelCItest

Examples

library(energy)
set.seed(10)
# x and y dependent, but independent conditionally on z
z <- 10*runif(300)
x <- sin(z) + runif(300)
y <- cos(z) + runif(300)
plot(x,y)
hsic.gamma(x,y)
hsic.perm(x,y)
dcov.test(x,y)
hsic.clust(x,y,z)

hsic.gamma

Hilbert Schmidt Independence Criterion gamma test

Description

Test to check the independence between two variables x and y using HSIC. The hsic.gamma() function, uses Hilbert-Schmidt independence criterion to test for independence between random variables.

Usage

hsic.gamma(x, y, sig = 1, numCol = 100)

Arguments

x data of first sample
y data of second sample
sig Gaussian kernel width for HSIC tests. Default is 1
numCol maximum number of columns that we use for the incomplete Cholesky decomposition
hsic.gamma

Details
Let \( x \) and \( y \) be two samples of length \( n \). Gram matrices \( K \) and \( L \) are defined as:
\[
K_{i,j} = \exp\left(\frac{(x_i-x_j)^2}{\sigma^2}\right)
\]
and
\[
L_{i,j} = \exp\left(\frac{(y_i-y_j)^2}{\sigma^2}\right).
\]
Let \( H_{i,j} = \delta_{i,j} - \frac{1}{n} \). Let \( A = HKH \) and \( B = HLH \), then \( HSIC(x, y) = \frac{1}{n^2} Tr(AB) \). Gamma test compares \( HSIC(x, y) \) with the \( \alpha \) quantile of the gamma distribution with mean and variance such as HSIC under independence hypothesis.

Value
hsic.gamma() returns a list with class htest containing
- method: description of test
- statistic: observed value of the test statistic
- estimate: \( HSIC(x, y) \)
- estimates: a vector: \([HSIC(x, y), \text{mean of } HSIC(x, y), \text{variance of } HSIC(x, y)]\)
- replicates: replicates of the test statistic
- p.value: approximate p-value of the test
- data.name: description of data

Author(s)
Petras Verbyla (<petras.verbyla@mrc-bsu.cam.ac.uk>) and Nina Ines Bertille Desgranges

References

See Also
hsic.perm, hsic.clust, kernelCItest

Examples
library(energy)
set.seed(10)
#independence
x <- runif(300)
y <- runif(300)
hsic.gamma(x,y)
hsic.perm(x,y)
dcov.gamma(x,y)
dcov.test(x,y)

#uncorrelated but not dependent
z <- 10*(runif(300)-0.5)
w <- z^2 + 10*runif(300)
cor(z,w)
hsic.gamma(z,w)
hsic.perm(z,w)
dcov.gamma(z,w)
dcov.test(z,w)

hsic.perm

*Hilber Schmidt Independence Criterion permutation test*

**Description**

Test to check the independence between two variables \(x\) and \(y\) using HSIC. The `hsic.perm()` function, uses Hilbert-Schmidt independence criterion to test for independence between random variables.

**Usage**

```r
hsic.perm(x, y, sig = 1, p = 100, numCol = 50)
```

**Arguments**

- `x`: data of first sample
- `y`: data of second sample
- `sig`: Gaussian kernel width for HSIC tests. Default is 1
- `p`: Number of permutations. Default is 100
- `numCol`: maximum number of columns that we use for the incomplete Cholesky decomposition

**Details**

Let \(x\) and \(y\) be two samples of length \(n\). Gram matrices \(K\) and \(L\) are defined as:

\[
K_{i,j} = \exp \left( \frac{(x_i - x_j)^2}{\sigma^2} \right)
\]

and

\[
L_{i,j} = \exp \left( \frac{(y_i - y_j)^2}{\sigma^2} \right). \]

Let

\[
H_{i,j} = \delta_{i,j} - \frac{1}{n}.
\]

Let \(A = HKH\) and \(B = HLH\), then

\[
HSIC(x, y) = \frac{1}{n} Tr(AB).
\]

Permutation test permutes \(y\) \(p\) times to get \(y_{(p)}\) and calculates

\[
HSIC(x, y_{(p)}).
\]

**Value**

`hsic.perm()` returns a list with class `htest` containing

- `method`: description of test
- `statistic`: observed value of the test statistic
- `estimate`: \(HSIC(x,y)\)
- `estimates`: a vector: [\(HSIC(x,y)\), mean of \(HSIC(x,y)\), variance of \(HSIC(x,y)\)]
- `replicates`: replicates of the test statistic
- `p.value`: approximate p-value of the test
- `data.name`: description of data
**Author(s)**

Petras Verbyla (<petras.verbyla@mrc-bsu.cam.ac.uk>) and Nina Ines Bertille Desgranges

**References**


**See Also**

hsic.gamma, hsic.clust, kernelCltest

**Examples**

```r
library(energy)
set.seed(10)
#independence
x <- runif(300)
y <- runif(300)
hsic.gamma(x,y)
hsic.perm(x,y)
dcov.gamma(x,y)
dcov.test(x,y)

#uncorrelated but not dependent
z <- 10*(runif(300)-0.5)
w <- z^2 + 10*runif(300)

cor(z,w)
hsic.gamma(z,w)
hsic.perm(z,w)
dcov.gamma(z,w)
dcov.test(z,w)
```

---

**Description**

Test to check the independence between two variables x and y using HSIC. The hsic.test() function, uses Hilbert-Schmidt independence criterion to test for independence between two random variables.

**Usage**

```r
hsic.test(x, y, p = 0, hsic.method = c("gamma", "perm"), sig = 1,
          numCol = floor(length(x)/10))
```
Arguments

x  data of first sample
y  data of second sample
p  number of replicates, if 0
hsic.method  method for HSIC test, either gamma test \texttt{hsic.gamma} or permutation test \texttt{hsic.perm}
sig  Gaussian kernel width for HSIC. Default is 1
numCol  number of columns in the Incomplete Cholesky Decomposition of Gram matrices. Default is floor(length(x)/10)

Details

Let x and y be two samples of length \( n \). Gram matrices K and L are defined as: 

\[
K_{i,j} = \exp\left(\frac{(x_i - x_j)^2}{\sigma^2}\right)
\]

and 

\[
L_{i,j} = \exp\left(\frac{(y_i - y_j)^2}{\sigma^2}\right).
\]

Let 

\[
H_{i,j} = \delta_{i,j} - \frac{1}{n}.
\]

Let \( A = HKH \) and \( B = HLH \), then 

\[
HSIC(x, y) = \frac{1}{n^2} \text{Tr}(AB).
\]

Value

\texttt{hsic.gamma}() returns a list with class \texttt{htest} containing

- \texttt{method}  description of test
- \texttt{statistic}  observed value of the test statistic
- \texttt{estimate}  \( HSIC(x, y) \)
- \texttt{estimates}  a vector: \([HSIC(x, y), \text{mean of } HSIC(x, y), \text{variance of } HSIC(x, y)]\)
- \texttt{replicates}  replicates of the test statistic
- \texttt{p.value}  approximate p-value of the test
- \texttt{data.name}  description of data

Author(s)

Petras Verbyla (<petras.verbyla@mrc-bsu.cam.ac.uk>) and Nina Ines Bertille Desgranges

References


See Also

\texttt{hsic.perm}, \texttt{hsic.clust}, \texttt{kernelCItest}
Examples

library(energy)
set.seed(10)
# independence
x <- runif(300)
y <- runif(300)

hsic.gamma(x,y)
hsic.perm(x,y)
dcov.gamma(x,y)
dcov.test(x,y)

# uncorrelated but not dependent
z <- 10*(runif(300)-0.5)
w <- z^2 + 10*runif(300)

cor(z,w)
hsic.gamma(z,w)
hsic.perm(z,w)
dcov.gamma(z,w)
dcov.test(z,w)

kernelCItest  Kernel Conditional Independence test

Description

Test to check the (conditional) dependence between two variables x and y given a set of variables S, using independence criteria. The kernelCItest() function, uses Distance Covariance or Hilbert-Schmidt Independence Criterion to test for the (conditional) independence between random variables, with an interface that can easily by used in skeleton, pc or kpc.

Usage

kernelCItest(x, y, S = NULL, suffStat, verbose = FALSE, data,
ic.method = NULL, p = NULL, index = NULL, sig = NULL, numCol = NULL,
umCluster = NULL, eps = NULL, paral = NULL)

Arguments

x, y, S  It is tested, whether x and y are conditionally independent given the subset S of the remaining nodes. x, y, S all are integers, corresponding to variable or node numbers.
suffStat  a list of parameters consisting of data, ic.method, p, index, sig, numCol, numCluster, eps, paral
verbose  a logical parameter, if TRUE, detailed output is provided.
data  numeric matrix witch columns representing variables and rows representing samples
ic.method: Method for the (conditional) independence test: Distance Covariance (permutation or gamma test), HSIC (permutation or gamma test) or HSIC cluster.

p: Number of permutations for Distance Covariance, HSIC permutation and HSIC cluster tests. Default is Distance Covariance.

index: Number in (0,2] the power of the distance in the Distance Covariance.

sig: Gaussian kernel width for HSIC tests. Default is 1.

numCol: Number of columns used in the incomplete Cholesky decomposition. Default is 50.

numCluster: Number of clusters for kPC clust algorithm.

eps: Normalization parameter for kPC clust. Default is 0.1.

paral: Number of cores to use for parallel calculations.

Value

cornelCltest() returns the p-value of the test.

Author(s)

Petras Verbyla (<petras.verbyla@mrc-bsu.cam.ac.uk>) and Nina Ines Bertille Desgranges.

References


Examples

```
set.seed(10)
library(pcalg)
z <- 10*runif(300)
w <- 10*runif(300)
x <- sin(z) + runif(300)
y <- cos(z) + runif(300)
data <- cbind(x,y,z,w)

# conditionally independent

test1a <- kernelCltest(x=1,y=2,S=c(3),suffStat = list(data=data,ic.method="dcc.gamma"))
test2a <- kernelCltest(x=1,y=2,S=c(3),suffStat = list(data=data,ic.method="dcc.perm"))
test3a <- kernelCltest(x=1,y=2,S=c(3),suffStat = list(data=data,ic.method="hsic.gamma"))
test4a <- kernelCltest(x=1,y=2,S=c(3),suffStat = list(data=data,ic.method="hsic.perm"))
test5a <- kernelCltest(x=1,y=2,S=c(3),suffStat = list(data=data,ic.method="hsic.clust"))
test6a <- gaussCltest( x=1,y=2,S=c(3),suffStat = list(C=cor(data),n=4))
test1a
```
#dependent
test1b <- kernelCItest(x=1,y=2,S=c(4),suffStat = list(data=data,ic.method="dcc.gamma"))
test2b <- kernelCItest(x=1,y=2,S=c(4),suffStat = list(data=data,ic.method="dcc.perm"))
test3b <- kernelCItest(x=1,y=2,S=c(4),suffStat = list(data=data,ic.method="hsic.gamma"))
test4b <- kernelCItest(x=1,y=2,S=c(4),suffStat = list(data=data,ic.method="hsic.perm"))
test5b <- kernelCItest(x=1,y=2,S=c(4),suffStat = list(data=data,ic.method="hsic.clust"))
test6b <- gaussCItest( x=1,y=2,S=c(4),suffStat = list(C=cor(data),n=4))

test1b
test2b
test3b
test4b
test5b
test6b

---

**kpc**

*Estimate the WAN-PDAG using the kPC Algorithm*

**Description**

Estimates the weakly additive noise partially directed acyclic graph (WAN-PDAG) from observational data, using the kPC algorithm. This is a version of `pc` from pcalg package, that uses HSIC (`hsic.gamma`, `hsic.perm` or `hsic.clust`) or distance covariance (`dcov.test` or `dcov.gamma`) independence tests and `udag2wanpdag` instead of `udag2pdag` in the last step.

**Usage**

```r
kpc(suffStat, indepTest, alpha, labels, p, fixedGaps = NULL, fixedEdges = NULL, NAdelete = TRUE, m.max = Inf, u2pd = c("relaxed", "rand", "retry"), skel.method = c("stable", "original", "stable.fast"), conservative = FALSE, maj.rule = FALSE, solve.confl = FALSE, verbose = FALSE)
```

**Arguments**

- `suffStat`: a list of sufficient statistics, containing all necessary elements for the conditional independence decisions in the function `indepTest`
- `indepTest`: A function for testing conditional independence. It is internally called as `indepTest(x,y,S,suffStat)`, and tests conditional independence of x and y given S. Here, x and y are variables, and S is a (possibly empty) vector of variables (all variables are denoted by their column numbers in the adjacency matrix). `suffStat` is a list, see the argument above. The return value of `indepTest` is the p-value of the test for conditional independence. Default is `kernelCItest`.

---
alpha  significance level (number in (0,1) for the individual conditional independence tests.
labels  (optional) character vector of variable (or "node") names. Typically preferred to specifying p.
p  (optional) number of variables (or nodes). May be specified if labels are not, in which case labels is set to 1:p.
fixedGaps  A logical matrix of dimension p*p. If entry [i,j] or [j,i] (or both) are TRUE, the edge i-j is removed before starting the algorithm. Therefore, this edge is guaranteed to be absent in the resulting graph.
fixedEdges  A logical matrix of dimension p*p. If entry [i,j] or [j,i] (or both) are TRUE, the edge i-j is never considered for removal. Therefore, this edge is guaranteed to be present in the resulting graph.
NAdelete  If indepTest returns NA and this option is TRUE, the corresponding edge is deleted. If this option is FALSE, the edge is not deleted.
m.max  Maximal size of the conditioning sets that are considered in the conditional independence tests.
u2pd  String specifying the method for dealing with conflicting information when trying to orient edges (see details below).
skel.method  Character string specifying method; the default, "stable" provides an order-independent skeleton, see skeleton.
conservative  Logical indicating if the conservative PC is used. In this case, only option u2pd = "relaxed" is supported. Note that therefore the resulting object might not be extendable to a DAG. See details for more information.
maj.rule  Logical indicating that the triples shall be checked for ambiguity using a majority rule idea, which is less strict than the conservative PC algorithm. For more information, see details.
solve.conf1  If TRUE, the orientation of the v-structures and the orientation rules work with lists for candidate sets and allow bi-directed edges to resolve conflicting edge orientations. In this case, only option u2pd = relaxed is supported. Note, that therefore the resulting object might not be a CPDAG because bi-directed edges might be present. See details for more information.
verbose  If TRUE, detailed output is provided.

Details
For more information: pc.

Value
An object of class "pcAlgo" (see pcAlgo) containing an estimate of the equivalence class of the underlying DAG.

Author(s)
Petras Verbyla(<petras.verbyla@mrc-bsu.cam.ac.uk>)
References


Examples

```r
## Not run:
library(pcalg)
set.seed(4)
n <- 300
data <- NULL
x1 <- 2*(runif(n)-0.5)
x2 <- x1 + runif(n)-0.5
x3 <- x1^2 + 0.6*runif(n)
x4 <- rnorm(n)
x5 <- x3 + x4^2 + 2*runif(n)
x6 <- 10*(runif(n)-0.5)
x7 <- x6^2 + 5*runif(n)
x8 <- 2*x7^2 + 1.5*rnorm(n)
x9 <- x7 + 4*runif(n)
data <- cbind(x1,x2,x3,x4,x5,x6,x7,x8,x9)
true <- matrix(0,9,9)
true[c(1),c(2,3)]<-true[c(3,4),5]<-true[c(6),c(7)]<-true[c(7),c(8)]<-true[7,9]<-1
pc <- pc(suffStat = list(C = cor(data), n = 9),
         indepTest = gaussCItest,
         alpha = 0.9,
         labels = colnames(data),
         u2pd = "relaxed",
         skel.method = "stable",
         verbose = TRUE)
kpc1 <- kpc(suffStat = list(data=data, ic.method="dcc.perm"),
            indepTest = kernelCItest,
            alpha = 0.1,
            labels = colnames(data),
            u2pd = "relaxed",
            skel.method = "stable",
            verbose = TRUE)
kpc2 <- kpc(suffStat = list(data=data, ic.method="hsic.gamma"),
            indepTest = kernelCItest,
            alpha = 0.1,
            labels = colnames(data),
            u2pd = "relaxed",
            skel.method = "stable",
            verbose = TRUE)
kpc3 <- kpc(suffStat = list(data=data, ic.method="hsic.perm"),
            indepTest = kernelCItest,
            alpha = 0.1,
            labels = colnames(data),
            u2pd = "relaxed",
            skel.method = "stable",
            verbose = TRUE)
```

regrVonPS <- kpc(suffStat = list(data=data, ic.method="hsic.clust"),
    indepTest = kernelCItest,
    alpha = 0.1,
    labels = colnames(data),
    u2pd = "relaxed",
    skel.method = "stable",
    verbose = TRUE)

if (require(Rgraphviz)) {
    par(mfrow=c(2,3))
    plot(pc,main="pc")
    plot(kpc1,main="dpc.perm")
    plot(kpc2,main="kpc.gamma")
    plot(kpc3,main="kpc.perm")
    plot(kpc4,main="kpc.clust")
    plot(as(true,"graphNEL"),main="True DAG")
}

## End(Not run)

### Description

Uses the generalised additive model `gam` to non-linearly and non-parametrically regress variable V on its parents and set of variables S.

### Usage

```r
regrVonPS(G, V, S, suffStat, indepTest = kernelCItest, alpha = 0.2)
```

### Arguments

- **G**: adjacency matrix, for the graph
- **V**: integer, node which we regress
- **S**: integer(s), set we regress on
- **suffStat**: sufficient statistics to perform the independence test `kernelCItest`
- **indepTest**: independence test to check for dependence between residuals of V and S
- **alpha**: numeric cutoff for significance level of individual partial correlation tests

### Value

`regrVonPS()` returns the number of p-values smaller than the cutoff, i.e 0 means residuals of V are independent of all variables in S

### Author(s)

Petras Verbyla (<petras.verbyla@mrc-bsu.cam.ac.uk>)
Description

Uses the generalised additive model `gam` to non-linearly and non-parametrically regress set of variables $X$ on a set of variables $S$ and returns residuals of $X$.

Usage

```r
regrXonS(X, S)
```

Arguments

- **X**: numeric matrix, set of variables to be regressed. Each column represents separate variable
- **S**: numeric matrix, set of variables we will regress on. Each column represents separate variable

Details

If the number of variables in $S$ is $\leq 5$ we use `frml.full.smooth` as formula for `gam` to regress $X$ on $S$, otherwise we use `frml.additive.smooth`.

Value

`regrXonS()` returns the residuals of $X$ regressed on $S$.

Author(s)

Petras Verbyla (<petras.verbyla@mrc-bsu.cam.ac.uk>)

See Also

- `kernelCItest`

Examples

```r
set.seed(10)
library(energy)
z <- 10*runif(300)
w <- 10*runif(300)
x <- sin(z) + runif(300)
y <- cos(z) + runif(300)
data <- cbind(x,y,z,w)

hsic.gamma(x,y)
hsic.perm(x,y)
dcov.test(x,y)
```
resid <- regrXonS(cbind(x,y),cbind(z,w))

hsic.gamma(resid[,1],resid[,2])
hsic.perm(resid[,1],resid[,2])
dcov.test(resid[,1],resid[,2])

udag2wanpdag

---

**Last kPC Algorithm Step: Extend Object with Skeleton to Completed PDAG**

**Description**

This function performs the last (generalised transitive) step in the kpc algorithm. It transforms an object of the class "pcAlgo" containing a skeleton and corresponding conditional independence information into a weakly additive noise directed acyclic graph (CPDAG). The functions first determine the v-structures in the collider step, and then performs the Generalised Transitive Step as described in Tillman et al (2009) to orient as many of the remaining edges as possible.

**Usage**

```r
udag2wanpdag(gInput, suffStat, indepTest = kernelCItest, alpha = 0.2,
 verbose = FALSE, unfVect = NULL, solve.confl = FALSE,
 orientCollider = TRUE, rules = rep(TRUE, 3))
```

**Arguments**

- **gInput**: "pcAlgo"-object containing skeleton and conditional independence information.
- **suffStat**: a list of sufficient statistics, containing all necessary elements for the conditional independence decisions in the function indepTest.
- **indepTest**: A function for testing conditional independence. It is internally called as indepTest(x,y,S,suffStat). Default is kernelCItest.
- **alpha**: significance level (number in (0,1) for the individual conditional independence tests.
- **verbose**: 0: No output; 1: Details
- **unfVect**: vector containing numbers that encode ambiguous triples (as returned by pc.cons.intern). This is needed in the conservative and majority rule PC algorithms.
- **solve.confl**: if TRUE, the orientation of the v-structures and the orientation rules work with lists for candidate sets and allow bi-directed edges to resolve conflicting edge orientations. Note that therefore the resulting object is order-independent but might not be a PDAG because bi-directed edges can be present.
- **orientCollider**: if TRUE, collider are oriented.
- **rules**: Array of length 3 containing TRUE or FALSE for each rule. TRUE in position i means that rule i (Ri) will be applied. By default, all rules are used.
- **gInput**
Details

First we perform a collider step, that is orienting triples a-b-c as a->b<-c iff b is not in separating set of a and c. Then we orient edges a-S as a->S if b_r is independent of a set S, where b_r are the residuals of b non parametrically regressed on S and parents of b and none of the edges S_i-a can be oriented as S_i->a, that is residuals S_i_r would be independent of a.

Value

An oriented object of class "pcAlgo".

References


Examples

```r
## Not run:
library(pcalg)
set.seed(4)
n <- 300
data <- NULL
x1 <- 2*(runif(n)-0.5)
x2 <- x1 + runif(n)-0.5
x3 <- x1^2 + 0.6*runif(n)
x4 <- rnorm(n)
x5 <- x3 + x4^2 + 2*runif(n)
x6 <- 10*(runif(n)-0.5)
x7 <- x6^2 + 10*runif(n)
x8 <- 2*x7^2 + rnorm(n)
x9 <- x7 + 5*runif(n)
data <- cbind(x1,x2,x3,x4,x5,x6,x7,x8,x9)
true <- matrix(0,9,9)
true[c(1),c(2,3)]<-true[c(3,4),5]<-true[c(6),c(7)]<-true[c(7),c(8)]<-true[7,9]<-1
## estimate skeleton
resU1 <- skeleton(suffStat = list(data=data, ic.method="dcc.perm", p=200),
indepTest = kernelCItest,
verbose = TRUE, alpha = 0.1, p=9)
resU2 <- skeleton(suffStat = list(data=data, ic.method="hsic.gamma",
sig=1, numCol = 50),
indepTest = kernelCItest,
verbose = TRUE, alpha = 0.1, p=9)
resU3 <- skeleton(suffStat = list(data=data, ic.method="hsic.perm",
sig=1, numCol = 50, p=200),
indepTest = kernelCItest,
verbose = TRUE, alpha = 0.1, p=9)
resU4 <- skeleton(suffStat = list(data=data, ic.method="hsic.clust",
p=200, sig=1, numCluster=100, numCol = 50),
```
udag2wanpdag

```r
eps = 0.1, paral = 1),
indepTest = kernelCItest,
verbose = TRUE, alpha = 0.1, p=9)

resU5 <- skeleton(suffStat = list(C = cor(data), n = n),
indepTest = gaussCItest,
verbose = TRUE, alpha = 0.1, p=9)

if (require(Rgraphviz)) {
  par(mfrow=c(2,3))
  plot(resU1,main="dpc")
  plot(resU2,main="kpc-resid-gamma")
  plot(resU3,main="kpc-resid-perm")
  plot(resU4,main="kpc-clust")
  plot(resU5,main="pc")
  plot(as(true,"graphNEL"),main="True DAG")
}

## orient edges using three different methods
resD1 <- udag2wanpdag(gInput = resU1,
suffStat = list(data=data, ic.method="dcc.perm", sig=1, numCol = 50, p=200),
indepTest = kernelCItest,
verbose = TRUE, alpha = 0.1)
resD2 <- udag2wanpdag(gInput = resU1,
suffStat = list(data=data, ic.method="hsic.gamma", sig=1, numCol = 50),
indepTest = kernelCItest,
verbose = TRUE, alpha = 0.1)
resD3 <- udag2wanpdag(gInput = resU1,
suffStat = list(data=data, ic.method="hsic.perm", sig=1, numCol = 50, p=200),
indepTest = kernelCItest,
verbose = TRUE, alpha = 0.1)
resD4 <- udag2pdagRelaxed(gInput = resU1, verbose = T)
if (require(Rgraphviz)) {
  par(mfrow=c(2,3))
  plot(resD1,main="dpc")
  plot(resD2,main="kpc-resid-gamma")
  plot(resD3,main="kpc-resid-perm")
  plot(resD4,main="pc")
  plot(as(true,"graphNEL"),main="True DAG")
}

## End(Not run)
```
Index

dcov.gamma, 2, 13
dcov.test, 2, 3, 13

frml.additive.smooth, 3, 17
frml.full.smooth, 4, 17

gam, 3, 4, 16, 17

hsic.clust, 3, 5, 7, 9, 10, 13
hsic.gamma, 3, 6, 6, 9, 10, 13
hsic.perm, 3, 6, 7, 8, 10, 13
hsic.test, 9

kernelCItest, 3, 6, 7, 9, 10, 11, 13, 16–18
kpc, 11, 13, 18

list, 13

pc, 11, 13, 14
pcAlgo, 14

regrVonPS, 16
regrXonS, 3, 4, 17

skeleton, 11

udag2pdag, 13
udag2wanpdag, 13, 18