

Package ‘pcalg’

April 26, 2012

Version 1.1-5

Date 2012-04-26

Author Markus Kalisch, Martin Maechler, Diego Colombo

Maintainer Markus Kalisch <kalisch@stat.math.ethz.ch>

Title Estimation of CPDAG/PAG and causal inference using the IDA algorithm

Description Standard and robust estimation of the equivalence class of a Directed Acyclic Graph (DAG) via the PC-Algorithm. The equivalence class is represented by its (unique) Complete Partially Directed Acyclic Graph (CPDAG). Furthermore, a PAG instead of a CPDAG can be estimated if latent variables and/or selection variables are assumed to be present. FCI and RFCI are available for estimating PAGs. Functions for causal inference using the IDA algorithm (based on do-calculus of Judea Pearl) are provided for CPDAGs.

Depends methods, abind, corpcor, sfsmisc

Imports graph, RBGL, ggm, robustbase, graphics, vcd

Suggests MASS, Matrix, graph, Rgraphviz, ggm

License GPL (>= 2)

URL <http://pcalg.r-forge.r-project.org/>

Repository CRAN

Date/Publication 2012-04-26 17:34:25

R topics documented:

beta.special	3
beta.special.pcObj	4
binCItest	4
compareGraphs	5

condIndFisherZ	7
corGraph	9
dag2cpdag	10
disCItest	11
dsep	12
dsepTest	13
fci	14
fciAlgo-class	18
gaussCItest	20
getNextSet	21
gmB	23
gmD	23
gmG	24
gmI	25
gmL	26
gSquareBin	27
gSquareDis	29
ida	30
idaFast	33
mcor	35
pc	36
pc.cons.intern	40
pcAlgo	42
pcAlgo-class	43
pcorOrder	44
pcSelect	46
pcSelect.presel	48
pdag2dag	49
pdsep	50
plotAG	52
plotSG	53
qreach	54
randomDAG	55
rfci	56
rmvDAG	58
shd	60
skeleton	61
udag2pag	64
udag2pdag	66
udag2pdagRelaxed	67
udag2pdagSpecial	69

beta.special	<i>Compute set of intervention effects</i>
--------------	--

Description

This function is DEPRECATED! Use [ida](#) instead.

Usage

```
beta.special(dat=NA, x.pos, y.pos, verbose=0, a=0.01, myDAG=NA,
myplot=FALSE, perfect=FALSE, method="local", collTest=TRUE, pcObj=NA,
all.dags=NA, u2pd="rand")
```

Arguments

dat	data matrix
x.pos	Column of x in dat
y.pos	Column of y in dat
verbose	0=no comments, 2=detail on estimates
a	significance level of tests for finding CPDAG
myDAG	needed if true correlation matrix shall be computed
myplot	plot estimated graph
perfect	True cor matrix is calculated from myDAG
method	"local" - local (all combinations of parents in regr.); "global" - all DAGs
collTest	True - Exclude orientations of undirected edges that introduce a new collider
pcObj	Fit of PC Algorithm (CPDAG); if this is available, no new fit is done
all.dags	All DAGs in the format of function allDags; if this is available, no new function call allDags is done
u2pd	Function for converting udag to pdag; "rand": udag2pdag; "relaxed": udag2pdagRelaxed; "retry": udag2pdagSpecial

Value

estimates of intervention effects

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

`beta.special.pcObj` *Compute set of intervention effects in a fast way*

Description

This function is DEPRECATED! Use [ida](#) or [idaFast](#) instead.

Usage

```
beta.special.pcObj(x.pos, y.pos, pcObj, mcov=NA, amat=NA, amatSkel=NA, t.amat=NA)
```

Arguments

<code>x.pos</code>	Column of x in dat
<code>y.pos</code>	Column of y in dat
<code>pcObj</code>	Precomputed pc-object
<code>mcov</code>	covariance that was used in the pc-object fit
<code>amat, amatSkel, t.amat</code>	matrices that can be precomputed, if needed (see code for details on how to precompute)

Value

estimates of intervention effects

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

See Also

[pcAlgo](#), [dag2cpdag](#), [beta.special.pcObj](#)

`binCItest` *Test for (conditional) independence for binary data*

Description

This function tests for (conditional) independence between binary random variables. The function is written, so that it can be easily used in [skeleton](#), [pc](#) and [fci](#).

Usage

```
binCItest(x, y, S, suffStat)
```

Arguments

x	Position of variable X in the adjacency matrix
y	Position of variable Y in the adjacency matrix
S	Position of conditioning variables in the adjacency matrix
suffStat	A list with two elements: (1) Element "dm" containing the data matrix (columns are variables, rows are samples) and (2) element "adaptDF" as a boolean variable indicating whether to lower the degrees of freedom by one for each zero count. (The value for the degrees of freedom cannot go below 1.)

Details

This function is based on [gSquareBin](#); see its help file for details.

Value

The p-value of the test.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

See Also

[dsepTest](#), [gaussCItest](#) and [disCItest](#) for similar functions for a d-separation oracle, a conditional independence test for gaussian variables and a conditional independence test for discrete variables, respectively.

Examples

```
## Simulate data
set.seed(123)
x <- sample(1:2, 100, TRUE)
y <- sample(1:2, 100, TRUE)
z <- sample(1:2, 100, TRUE)
dat <- cbind(x,y,z)

suffStat <- list(dm = dat, adaptDF = FALSE)
binCItest(1,3,2,suffStat)
```

compareGraphs

Compare two graphs in terms of TPR, FPR and TDR

Description

Compares the true undirected graph with an estimated undirected graph in terms of True Positive Rate (TPR), False Positive Rate (FPR) and True Discovery Rate (TDR).

Usage

```
compareGraphs(gl, gt)
```

Arguments

gl	estimated graph (graph object)
gt	true graph (graph object)

Details

If the input graph is directed, the directions are omitted. Special cases:

- If the true graph contains no edges, the tpr is defined to be zero.
- Similarly, if the true graph contains no gaps, the fpr is defined to be one.
- If there are no edges in the true graph and there are none in the estimated graph, tdr is one. If there are none in the true graph but there are some in the estimated graph, tdr is zero.

Value

A named numeric vector with three numbers:

tpr	True Positive Rate: Number of correctly found edges (in estimated graph) divided by number of true edges (in true graph)
fpr	False Positive Rate: Number of incorrectly found edges divided by number of true gaps (in true graph)
tdr	True Discovery Rate: Number of correctly found edges divided by number of found edges (both in estimated graph)

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>) and Martin Maechler

See Also

[randomDAG](#) for generating a random DAG.

Examples

```
## generate a graph with 4 nodes
V <- LETTERS[1:4]
edL2 <- vector("list", length=4)
names(edL2) <- V
edL2[[1]] <- list(edges= 2)
edL2[[2]] <- list(edges= c(1,3,4))
edL2[[3]] <- list(edges= c(2,4))
edL2[[4]] <- list(edges= c(2,3))
gt <- new("graphNEL", nodes=V, edgeL=edL2, edgemode="undirected")

## change graph
```

```

gl <- graph::addEdge("A","C", gt,1)

## compare the two graphs
if (require(Rgraphviz)) {
  par(mfrow=c(2,1))
  plot(gt) ; title("True graph")
  plot(gl) ; title("Estimated graph")
  (cg <- compareGraphs(gl,gt))
}

```

condIndFisherZ

Conditional Independence by Fisher's Z-Transformation

Description

Using Fisher's z-transformation of the partial correlation, test for zero partial correlation for sets of normally distributed random variables.

Usage

```

condIndFisherZ(x, y, S, C, n, cutoff,
               verbose= isTRUE(getOption("verbose.pcalg.condIFz")))
zStat(x, y, S, C, n)

```

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.
C	correlation matrix of nodes
n	integer specifying the number of observations ("samples") used to estimate the correlation matrix C.
cutoff	numeric cutoff for significance level of individual partial correlation tests. Must be set to $qnorm(1 - \alpha/2)$ for a test significance level of α .
verbose	logical indicating whether some intermediate output should be shown (WARNING: This decreases the performance dramatically!)

Details

For gaussian random variables and after performing Fisher's z-transformation of the partial correlation, the test statistic `zStat()` is (asymptotically for large enough `n`) standard normally distributed. Partial correlation is tested in a two-sided hypothesis test, i.e., basically, `condIndFisherZ(*) == abs(zStat(*)) > qnorm(1 - alpha/2)`. In a multivariate normal distribution, zero partial correlation is equivalent to conditional independence.

Value

zStat() gives a number

$$Z = \sqrt{n - |S| - 3} \cdot \log((1 + r)/(1 - r))/2$$

which is asymptotically normally distributed under the null hypothesis of correlation 0.

condIndFisherZ() returns a **logical** *L* indicating whether the “*partial correlation of x and y given S is zero*” could not be rejected on the given significance level. More intuitively and for multivariate normal data, this means: If TRUE then it seems plausible, that x and y are conditionally independent given S. If FALSE then there was strong evidence found against this conditional independence statement.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>) and Martin Maechler

References

Markus Kalisch and Peter B\u00fchlmann (2005) *Estimating high-dimensional directed acyclic graphs with the PC-algorithm*; Research Report Nr.~130, ETH Zurich;
http://stat.ethz.ch/research/research_reports/2005

See Also

[pcorOrder](#) for computing a partial correlation given the correlation matrix in a recursive way.

Examples

```
set.seed(42)
## Generate four independent normal random variables
n <- 20
data <- matrix(rnorm(n*4),n,4)
## Compute corresponding correlation matrix
corMatrix <- cor(data)
## Test, whether variable 1 (col 1) and variable 2 (col 2) are
## independent given variable 3 (col 3) and variable 4 (col 4) on 0.05
## significance level
x <- 1
y <- 2
S <- c(3,4)
n <- 20
alpha <- 0.05
cutoff <- 1-qnorm(alpha/2)
(b1 <- condIndFisherZ(x,y,S,corMatrix,n,cutoff))
# -> 1 and 2 seem to be conditionally independent given 3,4

## Now an example with conditional dependence
data <- matrix(rnorm(n*3),n,3)
data[,3] <- 2*data[,1]
corMatrix <- cor(data)
(b2 <- condIndFisherZ(1,3,2,corMatrix,n,cutoff))
# -> 1 and 3 seem to be conditionally dependent given 2
```

`corGraph`*Computing the correlation graph*

Description

Computes the correlation graph. This is the graph in which an edge is drawn between node i and node j , if the null hypothesis “*Correlation between X_i and X_j is zero*” can be rejected at the given significance level α (*alpha*).

Usage

```
corGraph(dm, alpha=0.05, Cmethod="pearson")
```

Arguments

<code>dm</code>	Numeric matrix with rows as samples and columns as variables.
<code>alpha</code>	Significance level for correlation test (numeric)
<code>Cmethod</code>	A character string indicating which correlation coefficient is to be used for the test. One of "pearson", "kendall", or "spearman", can be abbreviated. (string)

Value

Undirected correlation graph (graph object)

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>) and Martin Maechler

Examples

```
## create correlated samples
x1 <- rnorm(100)
x2 <- rnorm(100)
mat <- cbind(x1,x2, x3 = x1+x2)

if (require(Rgraphviz)) {
  ## ‘analyze the data’
  (g <- corGraph(mat)) # a ‘graphNEL’ graph, undirected
  plot(g) # ==> (1) and (2) are each linked to (3)

  ## use different significance level and different method
  (g2 <- corGraph(mat, alpha=0.01, Cmethod="kendall"))
  plot(g2) ## same edges as ‘g’
}
```

`dag2cpdag`*Convert a DAG to a CPDAG*

Description

Convert a DAG to a Completed Partially Directed Acyclic Graph (CPDAG).

Usage

```
dag2cpdag(dag)
```

Arguments

`dag` DAG (graph object)

Details

This function converts a DAG (graph object) to its corresponding (unique) CPDAG (graph object), using the algorithm of Chickering (2002).

Value

A graph object containing the CPDAG.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

References

D.M. Chickering (2002), Learning Equivalence Classes of Bayesian-Network Structures, *Journal of Machine Learning Research* 2, 445-398.

See Also

[udag2pdag](#), [pdag2dag](#)

Examples

```
p <- 10 # number of random variables
s <- 0.4 # sparseness of the graph

## generate random data
set.seed(42)
g <- randomDAG(p,s) # generate a random DAG

res <- dag2cpdag(g)
```

disCItest	<i>Test for (conditional) independence for discrete data</i>
-----------	--

Description

This function tests for (conditional) independence between discrete random variables. The function is written in a way, so that it can be easily used in [skeleton](#), [pc](#) and [fci](#).

Usage

```
disCItest(x, y, S, suffStat)
```

Arguments

x	Position of variable X in the adjacency matrix
y	Position of variable Y in the adjacency matrix
S	Position of conditioning variables in the adjacency matrix
suffStat	A list with three elements: (1) Element "dm" containing the data matrix (columns are variables, rows are samples), (2) element "nlev" containing a vector with the numbers of levels for each variable and (3) element "adaptDF" as a boolean variable indicating whether to lower the degrees of freedom by one for each zero count. (The value for the degrees of freedom cannot go below 1.)

Details

This function is based on [gSquareDis](#); see its help file for details.

Value

The p-value of the test.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

See Also

[dsepTest](#), [gaussCItest](#) and [binCItest](#) for similar functions for a d-separation oracle, a conditional independence test for gaussian variables and a conditional independence test for binary variables, respectively.

Examples

```
## Simulate data
set.seed(123)
x <- sample(1:3,100,TRUE)
y <- sample(1:4,100,TRUE)
z <- sample(1:2,100,TRUE)
dat <- cbind(x,y,z)

suffStat <- list(dm = dat, nlev = c(3,4,2), adaptDF = FALSE)
disCItest(1,3,2,suffStat)
```

dsep

*Test for d-separation in a DAG***Description**

This function tests for d-separation of nodes in a DAG.

Usage

```
dsep(a, b, S, g, john.pairs = NA)
```

Arguments

a	Label (sic!) of node A
b	Label (sic!) of node B
S	Labels (sic!) of set of nodes on which is conditioned
g	Element "g" containing the Directed Acyclic Graph (object of <code>class</code> "graph", see graph-class from the package graph)
john.pairs	the shortest path distance matrix for all pairs of nodes as computed by johnson.all.pairs.sp from package RBGL .

Details

This function checks separation in the moralized graph as explained in Lauritzen (2004).

Value

TRUE if a and b are d-separated by S in G, otherwise FALSE.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

References

S.L. Lauritzen (2004), Graphical Models, *Oxford University Press*, Chapter 3.2.2

See Also

[dsepTest](#) for a wrapper of this function that can easily be included into [skeleton](#), [pc](#) or [fci](#)

Examples

```
## generate random DAG
p <- 8
set.seed(45)
myDAG <- randomDAG(p, prob = 0.3)
if (require(Rgraphviz)) {
  plot(myDAG)
}

## Examples for d-separation
dsep("1", "7", NULL, myDAG)
dsep("4", "5", NULL, myDAG)
dsep("4", "5", "2", myDAG)
dsep("4", "5", c("2", "3"), myDAG)

## Examples for d-connection
dsep("1", "3", NULL, myDAG)
dsep("1", "6", "3", myDAG)
dsep("4", "5", "8", myDAG)
```

dsepTest

Test for d-separation in a DAG

Description

This function tests for d-separation of nodes in a DAG. The function is written, so that it can easily be used in [skeleton](#), [pc](#), [fci](#).

Usage

```
dsepTest(x, y, S, suffStat)
```

Arguments

x	Position of variable X in the adjacency matrix
y	Position of variable Y in the adjacency matrix
S	Positions of conditioning variables in the adjacency matrix
suffStat	A list with two elements: (1) Element "g" containing the Directed Acyclic Graph (object of class "graph", see graph-class from the package graph) and (2) element "jp" containing the shortest path distance matrix for all pairs of nodes as computed by johnson.all.pairs.sp from package RBGL .

Details

The function is based on [dsep](#). For details on d-separation see the Lauritzen (2004).

Value

If x and y are d-separated by S in DAG G the result is 1, otherwise it is 0. This is analogous to the p-value of an ideal (without sampling error) conditional independence test on any distribution that is faithful to the DAG G .

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

References

S.L. Lauritzen (2004), Graphical Models, *Oxford University Press*.

See Also

[gaussCItest](#), [disCItest](#) and [binCItest](#) for similar functions for a conditional independence test for gaussian, discrete and binary variables, respectively.

Examples

```
p <- 8
set.seed(45)
myDAG <- randomDAG(p, prob = 0.3)

if (require(Rgraphviz)) {
  ## plot the DAG
  plot(myDAG, main = "randomDAG(10, prob = 0.2)")
}

## define sufficient statistics (d-separation oracle)
suffStat <- list(g = myDAG, jp = RBGL::johnson.all.pairs.sp(myDAG))

dsepTest(1,6, S= NULL, suffStat) ## not d-separated
dsepTest(1,6, S= 3, suffStat) ## not d-separated by node 3
dsepTest(1,6, S= c(3,4),suffStat) ## d-separated by node 3 and 4
```

fci

Estimate the equivalence class of a MAG (PAG) using the FCI Algorithm

Description

Estimate the equivalence class of a maximal ancestral graph (MAG) from observational data, using the FCI-algorithm.

Usage

```
fci(suffStat, indepTest, p, alpha, verbose = FALSE, fixedGaps = NULL,
    fixedEdges = NULL, NAdelate = TRUE, m.max = Inf, rules = rep(TRUE, 10),
    doPdsep = TRUE, conservative = c(FALSE, FALSE), biCC = FALSE,
    cons.rules = FALSE, labels = NA)
```

Arguments

suffStat	Sufficient statistics: List containing all necessary elements for the conditional independence decisions in the function <code>indepTest</code> .
indepTest	Predefined function for testing conditional independence. The function is internally called as <code>indepTest(x, y, S, suffStat)</code> , 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). <code>suffStat</code> is a list containing all relevant elements for the conditional independence decisions. The return value of <code>indepTest</code> is the p-value of the test for conditional independence.
p	Number of variables.
alpha	Significance level for the individual conditional independence tests.
verbose	If TRUE, detailed output is provided.
fixedGaps	A logical matrix of dimension $p \times p$. If entry <code>[i, j]</code> or <code>[j, i]</code> (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 $p \times p$. If entry <code>[i, j]</code> or <code>[j, i]</code> (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.
NAdelate	If <code>indepTest</code> 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.
rules	Logical vector of length 10 indicating which rules should be used when directing edges. The order of the rules is taken from Zhang (2009).
doPdsep	If TRUE, Possible-D-SEP is computed for all nodes, and all subsets of Possible-D-SEP are considered as conditioning sets in the conditional independence tests. If FALSE, Possible-D-SEP is not computed, so that the algorithm simplifies to the Modified PC algorithm of Spirtes, Glymour and Scheines (2000, page ...).
conservative	If the first argument is TRUE, the skeleton is computed in a conservative way. If the second argument is TRUE, the triples are checked for faithfulness again after possible deletion of edges when finding Possible-D-SEP. For more information, see Details.
biCC	If TRUE, only nodes on paths between a and c are considered to be in <code>sepset(a,c)</code> . Uses biconnected components.
cons.rules	If TRUE, an orientation rule that needs information on definite non-colliders is only applied, if the corresponding subgraph relevant for the rule does not involve an unfaithful triple.
labels	Vector of length p with node names. If NA, <code>as.character{1:p}</code> is used instead.

Details

This function is a generalization of the PC algorithm (see [pc](#)), in the sense that it allows arbitrarily many latent and selection variables. Under the assumption that the data are faithful to a DAG that includes all latent and selection variables, the FCI algorithm (Fast Causal Inference algorithm) estimates the equivalence class of MAGs that describe the conditional independence relationships between the observed variables.

We estimate an equivalence class of MAGs instead of DAGs, since DAGs are not closed under marginalization and conditioning (Richardson and Spirtes, 2002).

An equivalence class of a MAG can be uniquely represented by a partial ancestral graph (PAG). A PAG contains the following types of edges: o-o, o-, o->, ->, <->, -. The bidirected edges come from hidden variables, and the undirected edges come from selection variables. The edges have the following interpretation: (i) there is an edge between x and y if and only if variables x and y are conditionally dependent given S for all sets S consisting of all selection variables and a subset of the observed variables; (ii) a tail on an edge means that this tail is present in all MAGs in the equivalence class; (iii) an arrowhead on an edge means that this arrowhead is present in all MAGs in the equivalence class; (iv) a o-edgemark means that there is at least one MAG in the equivalence class where the edgemark is a tail, and at least one where the edgemark is an arrowhead. Information on the interpretation of edges in a MAG can be found in the references given below.

The first part of the FCI algorithm is analogous to the PC algorithm. It starts with a complete undirected graph and estimates an initial skeleton using the function [skeleton](#). All edges of this skeleton are of the form o-o. Due to the presence of hidden variables, it is no longer sufficient to consider only subsets of the neighborhoods of nodes x and y to decide whether the edge x - y should be removed. Therefore, the initial skeleton may contain some superfluous edges. These edges are removed in the next step of the algorithm. To decide whether edge x o-o y should be removed, one computes Possible-D-SEP(x) and Possible-D-SEP(y) and performs conditional independence tests of x and y given all possible subsets of Possible-D-SEP(x) and of Possible-D-SEP(y) (see helpfile of [pdsep](#)). Subsequently, the v -structures are determined (using information in `sepset`). Finally, as many as possible undetermined edge marks (o) are determined using (a subset of) the 10 orientation rules given by Zhang (2009).

The conservative FCI consists of two parts. In the first part (done if the first argument of `conservative` is TRUE), we call `pc.cons.internal` with option `version.unf = c(1,2)` after computing the skeleton. This is a slight variation of the conservative PC (which used `version.unf = c(2,2)`): If a is independent of c given some S in the skeleton (i.e., the edge a - c dropped out), but a and c remain dependent given all subsets of neighbors of either a or c , we will call all triples a - b - c 'faithful'. This is because in the FCI, the true separating set might be outside the neighborhood of either a or c . In the second part (done if the second argument of `conservative` is TRUE), we call `pc.cons.internal` with option `version.unf = c(1,2)` again after Possible-D-Sep was found and the graph potentially lost some edges. Therefore, new triples might have occurred. If this second part is done, the resulting information on `sepset` and faithful triples overwrites the previous and will be used for the subsequent orientation rules.

Value

An object of `class` `fciAlgo` (see [fciAlgo](#)) containing the estimated graph (in the form of an adjacency matrix with various possible edge marks), the conditioning sets that lead to edge removals (`sepset`) and several other parameters.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>) and Diego Colombo.

References

T.S. Richardson and P. Spirtes (2002). Ancestral graph Markov models. *Annals of Statistics* **30** 962-1030.

Markus Kalisch, Martin Maechler, Diego Colombo, Marloes H. Maathuis, Peter Buehlmann (2012). Causal Inference Using Graphical Models with the R Package pcalg. *Journal of Statistical Software* **47(11)** 1–26, <http://www.jstatsoft.org/v47/i11/>.

P. Spirtes, C. Glymour and R. Scheines (2000). *Causation, Prediction, and Search*, 2nd edition, MIT Press, Cambridge (MA).

P. Spirtes, C. Meek, T.S. Richardson (1999). In: *Computation, Causation and Discovery*. An algorithm for causal inference in the presence of latent variables and selection bias. Pages 211-252. MIT Press.

J. Zhang (2008). On the completeness of orientation rules for causal discovery in the presence of latent confounders and selection bias. *Artificial Intelligence* **172** 1873-1896.

See Also

[skeleton](#) for estimating a skeleton using the PC algorithm; [pc](#) for estimating a CPDAG using the PC algorithm; [pdsep](#) for computing Possible-D-Sep for each node and testing and adapting the graph accordingly; [qreach](#) for a fast way of finding possible d-sep for a given node; [gaussCItest](#), [disCItest](#), [binCItest](#) and [dsepTest](#) as examples for indepTest.

Examples

```
#####
## Example without latent variables
#####

set.seed(42)
p <- 7
## generate and draw random DAG :
myDAG <- randomDAG(p, prob = 0.4)
amat <- wgtMatrix(myDAG)
amat[amat!=0] <- 1
amat <- amat + t(amat)
amat[amat!=0] <- 1

myCPDAG <- dag2cpdag(myDAG)
amat2 <- t(wgtMatrix(myCPDAG))

mycor <- cov2cor(trueCov(myDAG))

## find skeleton and CPDAG using the FCI algorithm
suffStat <- list(C = cov2cor(trueCov(myDAG)), n = 10^9)
indepTest <- gaussCItest
res <- fci(suffStat, indepTest, p, alpha = 0.99, verbose=TRUE)
```

```
#####
## Example with hidden variables
## Zhang (2008), Fig. 6, p.1882
#####

## create the graph
p <- 4
amat1 <- t(matrix(c(0,1,0,0,1, 0,0,1,0,0, 0,0,0,1,0, 0,0,0,0,0, 0,0,0,1,0),5,5))
colnames(amat1) <- rownames(amat1) <- as.character(1:5)
L1 <- 1
V1 <- as.character(1:5)
edL1 <- vector("list",length=5)
names(edL1) <- V1
edL1[[1]] <- list(edges=c(2,4),weights=c(1,1))
edL1[[2]] <- list(edges=3,weights=c(1))
edL1[[3]] <- list(edges=5,weights=c(1))
edL1[[4]] <- list(edges=5,weights=c(1))
g1 <- new("graphNEL", nodes=V1, edgeL=edL1,edgemode="directed")

## compute the true covariance matrix of g1
cov.mat1 <- trueCov(g1)

## delete rows and columns belonging to latent variable L1
true.cov1 <- cov.mat1[-L1,-L1]

## transform covariance matrix into a correlation matrix
true.corr1 <- cov2cor(true.cov1)

## find PAG with FCI algorithm
## as dependence "oracle", we use the true correlation matrix in the
## function gaussCIttest with a large "virtual sample size" and a large
## alpha
suffStat1 <- list(C = true.corr1, n = 10^9)
indepTest1 <- gaussCIttest
true.pag1 <- fci(suffStat1, indepTest1, p, alpha = 0.99, verbose=TRUE)

## define PAG given in paper
corr.pag1 <- matrix(0,4,4)
corr.pag1[1,] <- c(0,1,1,0)
corr.pag1[2,] <- c(1,0,0,2)
corr.pag1[3,] <- c(1,0,0,2)
corr.pag1[4,] <- c(0,3,3,0)

## check if estimated and correct PAG are in agreement
all(corr.pag1==true.pag1@amat)
```

Description

This class of objects is returned by the function `fci` to represent the estimated PAG. Objects of this class have methods for the functions `plot`, `show` and `summary`.

Creation of objects

Objects can be created by calls of the form `new("fciAlgo", ...)`, but are typically the result of `fci(...)`.

Slots

`amat`: Object of class "matrix": The the estimated graph, represented by its adjacency matrix. The edge marks are encoded by numbers: 0 = no edge, 1 = circle, 2 = arrowhead, 3 = tail. If `amat[i,j] = 1` and `amat[j,i] = 2`, this represents the edge $i \rightarrow j$.

`call`: Object of class "call": the original function call

`n`: Object of class "integer": the sample size used to estimate the graph.

`max.ord`: Object of class "integer": the maximum size of the conditioning sets used in the conditional independence tests in the first part of the algorithm (i.e., in the function `skeleton`).

`n.edgetests`: Object of class "numeric": the number of conditional independence tests performed in the first part of the algorithm.

`sepset`: Object of class "list": the conditioning sets that led to edge deletions. The set that led to the removal of the edge $i-j$ is saved in either `sepset[[i]][[j]]` or `sepset[[j]][[i]]`.

`pMax`: Object of class "matrix": the (i,j) th entry of the matrix contains the maximum p-value of all conditional independence tests for edge $i-j$.

`allPdsep`: Object of class "list": the i th entry of this list contains Possible D-SEP of node i .

`n.edgetestsPDSEP`: Object of class "numeric": the number of new conditional independence tests (i.e., tests that were not done in the first part of the algorithm) that were performed while checking subsets of Possible D-SEP.

`max.ordPDSEP`: Object of class "integer": the maximum size of the conditioning sets used in the new conditional independence that were performed when checking subsets of Possible D-SEP.

Extends

Class `"gAlgo"`.

Methods

plot signature(`x = "fciAlgo"`): Plot the resulting graph

show signature(`object = "fciAlgo"`): Show basic properties of the fitted object

summary signature(`object = "fciAlgo"`): Show details of the fitted object

Author(s)

Markus Kalisch and Martin Maechler

See Also[fci](#), [pcAlgo](#)**Examples**

```
## look at slots of the class
showClass("fciAlgo")
## Not run:
## Suppose, fciObj is an object of class fciAlgo
## access slots by using the @ symbol
fciObj@amat ## adjacency matrix
fciObj@sepset ## separation sets

## use show, summary and plot method
show(fciObj)
summary(fciObj)
plot(fciObj)

## End(Not run)

## Also look at the extensive examples in ?fci !
```

gaussCItest

Test for (conditional) independence for gaussian data

Description

This function tests for (conditional) independence between gaussian random variables. The function is written, so that it can easily be used in [skeleton](#), [pc](#) and [fci](#).

Usage

```
gaussCItest(x, y, S, suffStat)
```

Arguments

x	Position of variable X in adjacency matrix
y	Position of variable Y in adjacency matrix
S	Position of the conditioning variables in the adjacency matrix
suffStat	A list with two elements: (1) Element "C" containing the correlation matrix of the data and (2) element "n" containing the sample size.

Details

The Fisher z transformation is used. See the help on [zStat](#) for details.

Value

The p-value of the test.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

See Also

[dsepTest](#), [discITest](#) and [binCITest](#) for similar functions for a d-separation oracle, a conditional independence test for discrete variables and a conditional independence test for binary variables, respectively.

Examples

```
## simulate data: x -> y -> z
set.seed(29)
x <- rnorm(100)
y <- 3*x + rnorm(100)
z <- 2*y + rnorm(100)
dat <- cbind(x,y,z)

## analyze data
suffStat <- list(C = cor(dat), n = nrow(dat))
gaussCITest(1,3,NULL,suffStat) ## dependent
gaussCITest(1,3,2,suffStat) ## independent
```

getNextSet

Iteration through a list of all combinations of choose(n,k)

Description

Given a combination of k elements out of the elements $1, \dots, n$, the next set of size k in a specified sequence is computed.

Usage

```
getNextSet(n,k,set)
```

Arguments

n	number of elements to choose from (integer)
k	size of chosen set (integer)
set	previous set in list (numeric vector)

Details

The initial set is 1:k. Last index varies quickest. Using the dynamic creation of sets reduces the memory demands dramatically for large sets. If complete lists of combination sets have to be produced and memory is no problem, the function `combn` from package **combinat** is an alternative.

Value

List with two elements:

<code>nextSet</code>	next set in list (numeric vector)
<code>wasLast</code>	logical indicating whether the end of the specified sequence is reached.

Author(s)

Markus Kalisch <kalisch@stat.math.ethz.ch> and Martin Maechler

See Also

This function is used in [skeleton](#).

Examples

```
## start from first set (1,2) and get the next set of size 2 out of 1:5
## notice that res$wasLast is FALSE :
str(r <- getNextSet(5,2,c(1,2)))

## input is the last set; notice that res$wasLast now is TRUE:
str(r2 <- getNextSet(5,2,c(4,5)))

## Show all sets of size k out of 1:n :
## {if you really want this in practice, use something like combn() !}
n <- 5
k <- 3
currentSet <- 1:k
(res <- rbind(currentSet, deparse.level = 0))
repeat {
  newEl <- getNextSet(n,k,currentSet)
  if (newEl$wasLast)
    break
  ## otherwise continue:
  currentSet <- newEl$nextSet
  res <- rbind(res, currentSet, deparse.level = 0)
}
res
stopifnot(choose(n,k) == nrow(res)) ## must be identical
```

gmB

Graphical Model 5-Dim Binary Example Data

Description

This data set contains a matrix containing information on five binary variables (coded as 0/1) and the corresponding DAG model.

Usage

```
data(gmB)
```

Format

The format is a list of two components

```
x: int [1:5000, 1:5] 0 1 1 0 0 1 1 0 1 1 ...
```

```
g: Formal class 'graphNEL' [package "graph"] with 6 slots
  ..@ nodes : chr [1:5] "1" "2" "3" "4" ...
  ..@ edgeL :List of 5
  .....
```

Details

The data was generated using Tetrad in the following way. A random DAG on five nodes was generated; binary variables were assigned to each node; then conditional probability tables corresponding to the structure of the generated DAG were constructed. Finally, 5000 samples were drawn using the conditional probability tables.

Examples

```
data(gmB)
## maybe str(gmB) ; plot(gmB) ...
```

gmD

Graphical Model Discrete 5-Dim Example Data

Description

This data set contains a matrix containing information on five discrete variables (levels are coded as numbers) and the corresponding DAG model.

Usage

```
data(gmD)
```

Format

The format is a list of two components

x: int [1:10000, 1:5] 2 2 1 1 1 2 2 0 2 0 ...

g: Formal class 'graphNEL' [package "graph"] with 6 slots
@ nodes : chr [1:5] "1" "2" "3" "4" ...
@ edgeL :List of 5

Details

The data was generated using Tetrad in the following way. A random DAG on five nodes was generated; discrete variables were assigned to each node (with 3, 2, 3, 4 and 2 levels); then conditional probability tables corresponding to the structure of the generated DAG were constructed. Finally, 10000 samples were drawn using the conditional probability tables.

Examples

```
data(gmD)
## maybe str(gmD) ; plot(gmD) ...
```

 gmG

Graphical Model 8-Dimensional Gaussian Example Data

Description

This data set contains a matrix containing information on eight gaussian variables and the corresponding DAG model.

Usage

```
data(gmG)
```

Format

The format is a list of two components

x: num [1:5000, 1:8] 0.181 0.354 0.504 0.701 -0.705 ...

g: Formal class 'graphNEL' [package "graph"] with 6 slots
@ nodes : chr [1:8] "1" "2" "3" "4" ...
@ edgeL :List of 8

Details

The data was generated as indicated below. First, a random DAG model was generated, then 5000 samples were drawn from this model.

Source

The data set is [identical](#) to the one generated by

```
## Used to generate "gmG"
set.seed(40)
p <- 8
n <- 5000
gGtrue <- randomDAG(p, prob = 0.3) ## true DAG
gmG <- list(x = rmvDAG(n, gGtrue), g = gGtrue)
```

Examples

```
data(gmG)
str(gmG, max=3)
pairs(gmG$x, gap = 0,
      panel=function(...) smoothScatter(..., add=TRUE))
```

gmI

Graphical Model IDA Data Example

Description

This data set contains a matrix containing information on seven gaussian variables and the corresponding DAG model.

Usage

```
data(gmI)
```

Format

The format is a list of two components

```
x: num [1:10000, 1:7] -1.727 1.69 0.504 2.528 0.549 ...
g: Formal class 'graphNEL' [package "graph"] with 6 slots
.. ..@ nodes : chr [1:7] "1" "2" "3" "4" ...
.. ..@ edgeL :List of 7
.....
```

Details

The data was generated as indicated below. First, a random DAG model was generated, then 10000 samples were drawn from this model.

Source

The data set is [identical](#) to the one generated by

```
## Used to generate "gmI"
set.seed(123)
p <- 7
n <- 10000
myDAG <- randomDAG(p, prob = 0.2) ## true DAG
datI <- rmvDAG(n, myDAG)
gmI <- list(x = datI, g = myDAG)
```

Examples

```
data(gmI)
str(gmI, max=3)
pairs(gmI$x, gap = 0,
      panel=function(...) smoothScatter(..., add=TRUE))
```

gmL

Latent Variable Graphical Model Data Example

Description

This data set contains a matrix containing information on four gaussian variables and the corresponding DAG model containing four observed and one latent variable.

Usage

```
data(gmL)
```

Format

The format is a list of 2 components

x: \$ x: num [1:10000, 1:4] 0.924 -0.189 1.016 0.363 0.497 attr(*, "dimnames")=List of 2 ..
 ..\$: NULL\$: chr [1:4] "2" "3" "4" "5"

g: \$ g:Formal class 'graphNEL' [package "graph"] with 6 slots@ nodes : chr [1:5] "1" "2" "3"
 "4"@ edgeL :List of 5

Details

The data was generated as indicated below. First, a random DAG model was generated with five nodes; then 10000 samples were drawn from this model; finally, variable one was declared to be latent and the corresponding column was deleted from the simulated data set.

Source

```
## Used to generate "gmL"
set.seed(47)
p <- 5
n <- 10000
gGtrue <- randomDAG(p, prob = 0.3) ## true DAG
myX <- rmvDAG(n, gGtrue)
colnames(myX) <- as.character(1:5)
gmL <- list(x = myX[,-1], g = gGtrue)
```

Examples

```
data(gmL)
str(gmL, max=3)

## the graph:
gmL$g
graph::nodes(gmL$g) ; str(graph::edges(gmL$g))
if(require("Rgraphviz"))
  plot(gmL$g, main = "gmL $ g -- latent variable example data")

pairs(gmL $x) # the data
```

gSquareBin

G square Test for (Conditional) Independence of Binary Data

Description

G^2 statistic to test for (conditional) independence of *binary* variables X and Y given the (possibly empty) set of binary variables S .

Usage

```
gSquareBin(x, y, S, dm, verbose = FALSE, adaptDF = FALSE)
```

Arguments

<code>x, y</code>	position (column number) of variable X (and Y respectively) in the adjacency matrix.
<code>S</code>	position of the conditioning variables in the adjacency set.
<code>dm</code>	data matrix (rows: samples, columns: variables) with binary entries
<code>verbose</code>	logical indicating if detailed output is to be provided.
<code>adaptDF</code>	lower the degrees of freedom by one for each zero count. The value for the degrees of freedom cannot go below 1.

Details

The G^2 statistic is used to test for (conditional) independence of X and Y given a set S (can be NULL). This function is a specialized version of [gSquareDis](#) which is for discrete variables with more than two levels.

Value

The p-value of the test.

Author(s)

Nicoletta Andri and Markus Kalisch (<kalisch@stat.math.ethz.ch>).

References

R.E. Neapolitan (2004). Learning Bayesian Networks. *Prentice Hall Series in Artificial Intelligence*. Chapter 10.3.1

See Also

[gSquareDis](#) for a (conditional) independence test for discrete variables with more than two levels.
[binCIttest](#) for a wrapper of this function that can be easily included in [skeleton](#), [pc](#) or [fci](#).

Examples

```
## Simulate data from a chain of 3 variables: x1 -> x2 -> x3
set.seed(123)
b0 <- 0
b1 <- 1
b2 <- 1
n <- 10000
x1 <- sample(c(0,1),n,replace=TRUE)

## NB: plogis(u) := "expit(u)" := exp(u) / (1 + exp(u))
p2 <- plogis(b0 + b1*x1)

x2 <- numeric(length(x1))
for (i in 1:n) {
  x2[i] <- sample(c(0,1),1,prob=c(1-p2[i],p2[i]))
}
p3 <- plogis(b0 + b2*x2)
x3 <- numeric(length(x2))
for (i in 1:n) {
  x3[i] <- sample(c(0,1),1,prob=c(1-p3[i],p3[i]))
}

xtabs(~ x1+x2+x3)
dat <- cbind(x1,x2,x3)

## Test marginal and conditional independencies
gSquareBin(3,1,NULL,dat, verbose=TRUE)
```

```

gSquareBin(3,1, 2, dat)
gSquareBin(1,3, 2, dat) # the same
gSquareBin(1,3, 2, dat, adaptDF=TRUE)

```

gSquareDis

G square Test for (Conditional) Independence of Discrete Data

Description

G^2 statistic to test for (conditional) independence of *discrete* variables X and Y given the (possibly empty) set of discrete variables S .

Usage

```
gSquareDis(x, y, S, dm, nlev, verbose = FALSE, adaptDF = FALSE)
```

Arguments

<code>x,y</code>	position (column number) of variable X (and Y respectively) in the adjacency matrix.
<code>S</code>	position of the conditioning variables in the adjacency set.
<code>dm</code>	data matrix (rows: samples, columns: variables) with binary entries
<code>nlev</code>	vector with numbers of levels for each variable
<code>verbose</code>	logical indicating if detailed output is to be provided.
<code>adaptDF</code>	lower the degrees of freedom by one for each zero count. The value for the degrees of freedom cannot go below 1.

Details

The G^2 statistic is used to test for (conditional) independence of X and Y given a set S (can be NULL). If only binary variables are involved, [gSquareBin](#) is a specialized alternative to this function.

Value

The p-value of the test.

Author(s)

Nicoletta Andri and Markus Kalisch (<kalisch@stat.math.ethz.ch>).

References

R.E. Neapolitan (2004). Learning Bayesian Networks. *Prentice Hall Series in Artificial Intelligence*. Chapter 10.3.1

See Also

[gSquareBin](#) for a (conditional) independence test for binary variables. [disCItest](#) for a wrapper of this function that can be easily included in [skeleton](#), [pc](#) or [fci](#).

Examples

```
## Simulate data
x <- sample(1:3,100,TRUE)
y <- sample(1:4,100,TRUE)
z <- sample(1:2,100,TRUE)
dat <- cbind(x,y,z)

## Analyze data
gSquareDis(1,3,2,dat,nlev = c(3,4,2))
gSquareDis(1,3,2,dat,nlev = c(3,4,2), verbose=TRUE, adaptDF=TRUE)
```

ida

Estimate Multiset of Possible Total Causal Effects

Description

This function estimates the multiset of possible total causal effects of one variable (x) onto another variable (y) from observational data.

Usage

```
ida(x.pos, y.pos, mcov, graphEst, method="local",
    y.notparent = FALSE, verbose=FALSE, all.dags = NA)

causalEffect(g, y, x)
```

Arguments

<code>x.pos, x</code>	Position of variable x in the covariance matrix.
<code>y.pos, y</code>	Position of variable y in the covariance matrix.
<code>mcov</code>	Covariance matrix that was used to estimate <code>graphEst</code> .
<code>graphEst</code>	Estimated CPDAG from the function pc . If the output of pc is <code>pc.fit</code> , then the estimated CPDAG can be obtained by <code>pc.fit@graph</code> .
<code>method</code>	If <code>method="global"</code> , the algorithm considers all DAGs in the equivalence class of the CPDAG (slow). If <code>method="local"</code> , the algorithm only considers the local neighborhood of x in the CPDAG (fast). See details below.
<code>y.notparent</code>	If <code>y.notparent=TRUE</code> , any edge between x and y is assumed to be of the form $x \rightarrow y$.
<code>verbose</code>	If <code>TRUE</code> , details on the regressions are given.

<code>all.dags</code>	All DAGs in the equivalence class of the CPDAG can be precomputed by the function <code>allDags</code> and passed via this argument. If this is done, <code>allDags(. .)</code> is not called internally. This option is only relevant when using <code>method="global"</code> .
<code>g</code>	graph in which the causal effect is sought.

Details

It is assumed that we have observational data that are multivariate Gaussian and faithful to the true (but unknown) underlying causal DAG (without hidden variables). Under these assumptions, this function estimates the multiset of possible total causal effects of x on y , where the total causal effect is defined via Pearl's do-calculus as $E(Y|\text{do}(X=z+1)) - E(Y|\text{do}(X=z))$ (this value does not depend on z , since Gaussianity implies that conditional expectations are linear).

We estimate a *set* of possible total causal effects instead of the unique total causal effect, since it is typically impossible to identify the latter when the true underlying causal DAG is unknown (even with an infinite amount of data). Conceptually, the method works as follows. First, we estimate the equivalence class of DAGs that describe the conditional independence relationships in the data, using the function `pc` (see the help file of this function). For each DAG G in the equivalence class, we apply Pearl's do-calculus to estimate the total causal effect of x on y . This can be done via a simple linear regression: if y is not a parent of x , we take the regression coefficient of x in the regression $\text{lm}(y \sim x + \text{pa}(x))$, where $\text{pa}(x)$ denotes the parents of x in the DAG G ; if y is a parent of x in G , we set the estimated causal effect to zero.

If the equivalence class contains k DAGs, this will yield k estimated total causal effects. Since we do not know which DAG is the true causal DAG, we do not know which estimated total causal effect of x on y is the correct one. Therefore, we return the entire multiset of k estimated effects (it is a multiset rather than a set because it can contain duplicate values).

One can take summary measures of the multiset. For example, the minimum absolute value provides a lower bound on the size of the true causal effect: if the minimum absolute value of all values in the multiset is larger than one, then we know that the size of the true causal effect (up to sampling error) must be larger than one.

If `method="global"`, the method as described above is carried out, where all DAGs in the equivalence class of the estimated CPDAG `graphEst` are computed using the function `allDags`. This method is suitable for small graphs (say, up to 10 nodes).

If `method="local"`, we do not determine all DAGs in the equivalence class of the CPDAG. Instead, we only consider the local neighborhood of x in the CPDAG. In particular, we consider all possible directions of undirected edges that have x as endpoint, such that no new v -structure is created. For each such configuration, we estimate the total causal effect of x on y as above, using linear regression.

At first sight, it is not clear that such a local configuration corresponds to a DAG in the equivalence class of the CPDAG, since it may be impossible to direct the remaining undirected edges without creating a directed cycle or a v -structure. However, Maathuis, Kalisch and Buehlmann (2009) showed that there is at least one DAG in the equivalence class for each such local configuration. As a result, it follows that the multisets of total causal effects of the "global" and the "local" method have the same unique values. They may, however, have different multiplicities. For example, a CPDAG may represent eight DAGs, and the global method may produce the multiset $\{1.3, -0.5, 0.7, 1.3, 1.3, -0.5, 0.7, 0.7\}$. The unique values in this set are $-0.5, 0.7$ and 1.3 , and the multiplicities are 2, 3 and 3. The local method, on the other hand, may yield $\{1.3, -0.5, -0.5, 0.7\}$. The unique values are $-0.5, 0.7$ and 1.3 , but the multiplicities are now 2, 1 and 1. The fact that the unique values

of the multisets of the "global" and "local" method are identical implies that summary measures of the multiset that only depend on the unique values (such as the minimum absolute value) can be estimate by the local method.

Value

A vector that represents the multiset containing the estimated possible total causal effects of x on y.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

References

M.H. Maathuis, M. Kalisch, P. Buehlmann (2009). Estimating high-dimensional intervention effects from observational data. *Annals of Statistics* **37**, 3133–3164.

M.H. Maathuis, D. Colombo, M. Kalisch, P. Buehlmann (2010). Predicting causal effects in large-scale systems from observational data. *Nature Methods*, to appear.

Markus Kalisch, Martin Maechler, Diego Colombo, Marloes H. Maathuis, Peter Buehlmann (2012). Causal Inference Using Graphical Models with the R Package pcalg. *Journal of Statistical Software* **47(11)** 1–26, <http://www.jstatsoft.org/v47/i11/>.

Pearl (2005). *Causality. Models, reasoning and inference*. Cambridge University Press, New York.

See Also

[pc](#) for estimating a CPDAG, [idaFast](#) for estimating the multiset of possible total causal effects for several target variables at the same time.

Examples

```
## Simulate the true DAG
set.seed(123)
p <- 7
myDAG <- randomDAG(p, prob = 0.2) ## true DAG
myCPDAG <- dag2cpdag(myDAG) ## true CPDAG
covTrue <- trueCov(myDAG) ## true covariance matrix

## simulate Gaussian data from the true DAG
n <- 10000
dat <- rmvDAG(n, myDAG)

## estimate CPDAG (see the help-file of the function "pc")
alpha <- 0.01
indepTest <- gaussCItest
suffStat <- list(C = cor(dat), n = n)
pc.fit <- pc(suffStat, indepTest, p, alpha)

if (require(Rgraphviz)) {
  ## plot the true and estimated graphs
  par(mfrow = c(1,2))
```

```

plot(myDAG, main = "True DAG")
plot(pc.fit, main = "Estimated CPDAG")
}

## Suppose that we know the true CPDAG and covariance matrix
ida(2,5,trueCov(myDAG),myCPDAG,method = "local")
ida(2,5,trueCov(myDAG),myCPDAG,method = "global")
## The global and local method produce the same unique values.

## From the true DAG, we can compute the true causal effect of 2 on 5
causalEffect(myDAG, 5, 2)
## Indeed, this value is contained in the values found by both the
## local and global method

## When working with data, we have to use the estimated CPDAG and
## the sample covariance matrix
ida(2,5,cov(dat),pc.fit@graph,method = "local")
ida(2,5,cov(dat),pc.fit@graph,method = "global")
## The unique values of the local and the global method are still identical,
## and the true causal effect is contained in both sets, up to a small
## estimation error (0.542 vs. 0.553).

```

idaFast

Multiset of Possible Total Causal Effects for Several Target Variables

Description

This function estimates the multiset of possible total causal effects of one variable (x) on a *several* (i.e., a vector of) target variables (y) from observational data.

`idaFast()` is more efficient than looping over `ida`. Only `method="local"` (see `ida`) is available.

Usage

```
idaFast(x.pos, y.pos.set, mcov, graphEst)
```

Arguments

<code>x.pos</code>	Position of variable x in the covariance matrix
<code>y.pos.set</code>	Vector containing the position of the target variables y in the covariance matrix
<code>mcov</code>	Covariance matrix that was used to estimate <code>graphEst</code>
<code>graphEst</code>	Estimated CPDAG from the function <code>pc</code> . If the output of <code>pc</code> is <code>pc.fit</code> , then the estimated CPDAG can be obtained by <code>pc.fit@graph</code> .

Details

This function performs `ida(x.pos, y.pos, mcov, graphEst, method="local", y.notparent=FALSE, verbose=FALSE)` for all values of `y.pos` in `y.pos.set` simultaneously, in an efficient way. See the help file of `ida` for more details. Note that the option `y.notparent = TRUE` is not implemented, since it is not clear how to do that efficiently without orienting all edges away from `y.pos.set` at the same time, which seems not to be desirable. Suggestions are welcome.

Value

Matrix with `length(y.pos.set)` rows. Row i contains the multiset of estimated possible total causal effects of x on $y.pos.set[i]$. Note that all multisets in the matrix have the same length, since the parents of x are the same for all elements of $y.pos.set$.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

References

M.H. Maathuis, M. Kalisch, P. Buehlmann (2009). Estimating high-dimensional intervention effects from observational data. *Annals of Statistics* **37**, 3122–3164.

Markus Kalisch, Martin Maechler, Diego Colombo, Marloes H. Maathuis, Peter Buehlmann (2012). Causal Inference Using Graphical Models with the R Package `pcalg`. *Journal of Statistical Software* **47(11)** 1–26, <http://www.jstatsoft.org/v47/i11/>.

M.H. Maathuis, D. Colombo, M. Kalisch, P. Buehlmann (2010). Predicting causal effects in large-scale systems from observational data. *Nature Methods*, to appear.

See Also

`pc` for estimating a CPDAG, and `ida` for estimating the multiset of possible total causal effects from observational data on only one target variable but with many more options (than here in `idaFast`).

Examples

```
## Simulate the true DAG
set.seed(123)
p <- 7
myDAG <- randomDAG(p, prob = 0.2) ## true DAG
myCPDAG <- dag2cpdag(myDAG) ## true CPDAG
covTrue <- trueCov(myDAG) ## true covariance matrix

## simulate data from the true DAG
n <- 10000
dat <- rmvDAG(n, myDAG)

## estimate CPDAG (see help on the function "pc")
alpha <- 0.01
indepTest <- gaussCItest
suffStat <- list(C = cor(dat), n = n)
pc.fit <- pc(suffStat, indepTest, p, alpha)

if(require(Rgraphviz))
  plot(myDAG)

(eff.est1 <- ida(2,5,cov(dat),pc.fit@graph,method="local",verbose=FALSE))
(eff.est2 <- ida(2,6,cov(dat),pc.fit@graph,method="local",verbose=FALSE))
(eff.est3 <- ida(2,7,cov(dat),pc.fit@graph,method="local",verbose=FALSE))
## These three computations can be combined in an efficient way
```

```
## by using idaFast :
(eff.estF <- idaFast(2,c(5,6,7),cov(dat),pc.fit@graph))
```

mcor

Compute (Large) Correlation Matrix

Description

Compute a correlation matrix, possibly by robust methods, applicable also for the case of a large number of variables.

Usage

```
mcor(dm, method = c("standard", "Qn", "QnStable",
                    "ogkScaleTau2", "ogkQn", "shrink"))
```

Arguments

dm	numeric matrix of data; rows are samples, columns are variables.
method	"standard" (default), "Qn", "QnStable", "ogkQn" and "shrink" invokes standard, elementwise robust (based on Q_n scale estimator, see Qn), robust (Qn using OGK, see covOGK) or shrunked () correlation estimate respectively.

Details

The "standard" method invokes a standard correlation estimator. "Qn" invokes a robust, element-wise correlation estimator based on the Qn scale estimate. "QnStable" also uses the Qn scale estimator, but uses an improved way of transforming that into the correlation estimator. "ogkQn" invokes a correlation estimator based on Qn using OGK. "shrink" is only useful when used with pcSelect. An optimal shrinkage parameter is used. Only correlation between response and covariates is shrunked.

Value

A correlation matrix estimated by the specified method.

Author(s)

Markus Kalisch <kalisch@stat.math.ethz.ch> and Martin Maechler

References

See those in the help pages for Qn and covOGK from package **robustbase**.

See Also

[Qn](#) and [covOGK](#) from package **robustbase**. [pcorOrder](#) for computing partial correlations.

Examples

```
## produce uncorrelated normal random variables
set.seed(42)
x <- rnorm(100)
y <- 2*x + rnorm(100)
## compute correlation of var1 and var2
mcor(cbind(x,y), method="standard")

## repeat but this time with heavy-tailed noise
yNoise <- 2*x + rcauchy(100)
mcor(cbind(x,yNoise), method="standard") ## shows almost no correlation
mcor(cbind(x,yNoise), method="Qn")      ## shows a lot correlation
mcor(cbind(x,yNoise), method="QnStable") ## shows still much correlation
mcor(cbind(x,yNoise), method="ogkQn")  ## ditto
```

 pc

Estimate the equivalence class of a DAG using the PC algorithm

Description

Estimate the equivalence class of a directed acyclic graph (DAG) from observational data, using the PC-algorithm.

Usage

```
pc(suffStat, indepTest, p, alpha, verbose = FALSE, fixedGaps = NULL,
   fixedEdges = NULL, NAdelate = TRUE, m.max = Inf, u2pd = "rand",
   conservative = FALSE)
```

Arguments

suffStat	Sufficient statistics: List containing all necessary elements for the conditional independence decisions in the function <code>indepTest</code> .
indepTest	Predefined function for testing conditional independence. The function is internally called as <code>indepTest(x,y,S,suffStat)</code> , 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). <code>suffStat</code> is a list containing all relevant elements for the conditional independence decisions. The return value of <code>indepTest</code> is the p-value of the test for conditional independence.
p	Number of variables.
alpha	Significance level for the individual conditional independence tests.
verbose	If TRUE, detailed output is provided.
fixedGaps	A logical matrix of dimension $p \times p$. 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.

<code>fixedEdges</code>	A logical matrix of dimension $p \times p$. 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.
<code>NAdelete</code>	If <code>indepTest</code> returns NA and this option is TRUE, the corresponding edge is deleted. If this option is FALSE, the edge is not deleted.
<code>m.max</code>	Maximal size of the conditioning sets that are considered in the conditional independence tests.
<code>u2pd</code>	Method for dealing with conflicting information when trying to orient edges (see details below).
<code>conservative</code>	If TRUE, the conservative PC is done. In this case, only option <code>u2pd = relaxed</code> is supported. Note, that therefore the resulting object might not be extendable to a DAG. See details for more information.

Details

Under the assumption that the distribution of the observed variables is faithful to a DAG, this function estimates the equivalence class of the DAG. We do not estimate the DAG itself, because this is typically impossible (even with an infinite amount of data), since different DAGs can describe the same conditional independence relationships. Since all DAGs in an equivalence class describe the same conditional independence relationships, they are equally valid ways to describe the conditional dependence structure that was given as input.

All DAGs in an equivalence class have the same skeleton (i.e., the same adjacency information) and the same v-structures (see definition below). However, the direction of some edges may be undetermined, in the sense that they point one way in one DAG in the equivalence class, while they point the other way in another DAG in the equivalence class.

An equivalence class can be uniquely represented by a completed partially directed acyclic graph (CPDAG). A CPDAG contains undirected and directed edges. The edges have the following interpretation: (i) there is a (directed or undirected) edge between i and j if and only if variables i and j are conditionally dependent given S for all possible subsets S of the remaining nodes; (ii) a directed edge $i \rightarrow j$ means that this directed edge is present in all DAGs in the equivalence class; (iii) an undirected edge $i-j$ means that there is at least one DAG in the equivalence class with edge $i \rightarrow j$ and there is at least one DAG in the equivalence class with edge $i < j$.

The CPDAG is estimated using the PC algorithm (named after its inventors **P**eter **S**irtes and **C**lark **G**lymour). The skeleton is estimated by the function `skeleton` (see the help file of this function for details). Subsequently, as many edges as possible are oriented. This is done in two steps.

First, the algorithm considers all triples (a, b, c) , where a and b are adjacent, b and c are adjacent, but a and c are not adjacent. For all such triples, we direct both edges towards b ($a \rightarrow b < c$) if and only if b was not part of the conditioning set that made the edge between a and c drop out. These conditioning sets were saved in `sepset`. The structure $a \rightarrow b < c$ is called a v-structure.

After determining all v-structures, there may still be undirected edges. It may be possible to direct some of these edges, since one can deduce that one of the two possible directions of the edge is invalid because it introduces a new v-structure or a directed cycle. Such edges are found by repeatedly applying rules R1-R3 as given in Algorithm 2 of Kalisch and Buhlmann (2007). The algorithm stops if none of the rules is applicable to the graph.

Sampling errors (or hidden variables) can lead to conflicting information about edge directions. For example, one may find that $a-b-c$ and $b-c-d$ should both be directed as v-structures. This gives

conflicting information about the edge b-c, since it should be directed as b<-c in v-structure a->b<-c, while it should be directed as b->c in v-structure b->c<-d. In such cases, we simply overwrite the directions of the conflicting edge. In the example above this means that we obtain a->b->c<-d if a-b-c was visited first, and a->b<-c<-d if b-c-d was visited first.

Sampling errors or hidden variables can also lead to invalid CPDAGs, meaning that there does not exist a DAG that has the same skeleton and v-structures as the graph found by the algorithm. An example of this is an undirected cycle consisting of the edges a-b-c-d and d-a. In this case it is impossible to direct the edges without creating a cycle or a new v-structure. The option `u2pd` specifies what should be done in such a situation. If the option is set to "relaxed", the algorithm simply outputs the invalid CPDAG. If the option is set to "rand", all direction information is discarded and a random DAG is generated on the skeleton. If the option is set to "retry", up to 100 combinations of possible directions of the ambiguous edges are tried, and the first combination that results in a valid CPDAG is chosen. If no valid combination is found, an arbitrary DAG is generated on the skeleton as in the option "rand".

The conservative PC algorithm (`"conservative = TRUE"`) is a slight variation of the PC algorithm. After the skeleton is computed, all potential v-structures a-b-c are checked in the following way. We test whether a and c are independent conditioning on any subset of the neighbors of a or any subset of the neighbors of c. If b is in no such conditioning set (and not in the original sepset - ???DIEGO: 2nd argument, version 2) or in all such conditioning sets (and in the original sepset), no further action is taken and the usual PC is continued. If, however, b is in only some conditioning sets, the triple a-b-c is marked 'unfaithful'. Moreover, if in the conservative step there is no subset among the neighbors that makes a and c independent, the triple is marked 'unfaithful' (???DIEGO: 1st argument, version 2). An unfaithful triple is not oriented as a v-structure. Furthermore, no later orientation rule that needs to know whether a-b-c is a v-structure or not is applied. (The internal function `pc.cons.internal` will be called with `version.unf = c(2,2)`.)

Notes: (1) Throughout, the algorithm works with the column positions of the variables in the adjacency matrix, and not with the names of the variables. (2) When plotting the object, undirected and bidirected edges are equivalent.

Value

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

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>) and Martin Maechler.

References

P. Spirtes, C. Glymour and R. Scheines (2000). *Causation, Prediction, and Search*, 2nd edition. The MIT Press.

Markus Kalisch, Martin Maechler, Diego Colombo, Marloes H. Maathuis, Peter Buehlmann (2012). Causal Inference Using Graphical Models with the R Package `pcalg`. *Journal of Statistical Software* **47(11)** 1–26, <http://www.jstatsoft.org/v47/i11/>.

M. Kalisch and P. Buehlmann (2007). Estimating high-dimensional directed acyclic graphs with the PC-algorithm. *JMLR* **8** 613-636.

See Also

[skeleton](#) for estimating a skeleton of a DAG; [udag2pdag](#) for converting the skeleton to a CPDAG; [gaussCItest](#), [disCItest](#), [binCItest](#) and [dsepTest](#) as examples for `indepTest`.

Examples

```
#####
## Using Gaussian Data
#####
## Load predefined data
data(gmG)
n <- nrow(gmG$x)
p <- ncol(gmG$x)

## define independence test (partial correlations)
indepTest <- gaussCItest
## define sufficient statistics
suffStat <- list(C = cor(gmG$x), n = n)
## estimate CPDAG
alpha <- 0.01
pc.fit <- pc(suffStat, indepTest, p, alpha, verbose = TRUE)
if (require(Rgraphviz)) {
  ## show estimated CPDAG
  par(mfrow=c(1,2))
  plot(pc.fit, main = "Estimated CPDAG")
  plot(gmG$g, main = "True DAG")
}
#####
## Using d-separation oracle
#####
## define independence test
indepTest <- dsepTest
## define sufficient statistics (d-separation oracle)
suffStat <- list(g = gmG$g, jp = RBGL::johnson.all.pairs.sp(gmG$g))
## estimate CPDAG
alpha <- 0.01 ## value is irrelevant as dsepTest returns either 0 or 1
fit <- pc(suffStat, indepTest, p, alpha)
if (require(Rgraphviz)) {
  ## show estimated CPDAG
  plot(fit, main = "Estimated CPDAG")
  plot(gmG$g, main = "True DAG")
}
#####
## Using discrete data
#####
## Load data
data(gmD)
p <- ncol(gmD$x)
## define independence test (G^2 statistics)
indepTest <- disCItest
## define sufficient statistics
suffStat <- list(dm = gmD$x, nlev = c(3,2,3,4,2), adaptDF = FALSE)
```

```

## estimate CPDAG
alpha <- 0.01
pc.fit <- pc(suffStat, indepTest, p, alpha, verbose = TRUE)
if (require(Rgraphviz)) {
  ## show estimated CPDAG
  par(mfrow = c(1,2))
  plot(pc.fit, main = "Estimated CPDAG")
  plot(gmD$g, main = "True DAG")
}

#####
## Using binary data
#####
## Load binary data
data(gmB)
p <- ncol(gmB$x)
## define independence test
indepTest <- binCItest
## define sufficient statistics
suffStat <- list(dm = gmB$x, adaptDF = FALSE)
## estimate CPDAG
alpha <- 0.01
pc.fit <- pc(suffStat, indepTest, p, alpha, verbose = TRUE)
if (require(Rgraphviz)) {
  ## show estimated CPDAG
  plot(pc.fit, main = "Estimated CPDAG")
  plot(gmB$g, main = "True DAG")
}

```

pc.cons.intern

Internal function for conservative PC algorithm

Description

This function is not intended for direct usage. Use `pc` or `fci` using options for the conservative version of the algorithms instead.

For any unshielded triple A-B-C, consider all subsets D of the neighbors of A and of the neighbors of C, and record the sets D for which A and C are conditionally independent given D. If B is in none of these sets, do nothing (it is a v-structure). If B is in all sets, do nothing (it is not a v-structure). If B is in some but not all sets, mark the triple as “unfaithful”.

Usage

```

pc.cons.intern(sk, suffStat, indepTest, alpha, verbose = FALSE,
              version.unf = c(NA, NA))

```

Arguments

sk	an object as returned from skeleton() .
suffStat	sufficient statistic: List containing all necessary elements for the conditional independence decisions in the function <code>indepTest</code> .
indepTest	predefined function for testing conditional independence. The function is internally called as <code>indepTest(x, y, S, suffStat)</code> , 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). <code>suffStat</code> is a list containing all relevant elements for the conditional independence decisions. The return value of <code>indepTest</code> is the p-value of the test for conditional independence.
alpha	significance level for the individual conditional independence tests.
verbose	If TRUE, detailed output is provided.
version.unf	vector of length two. First argument: Consider the case, where a indep c given S in the skeleton; furthermore, suppose that a and c are dependent given every subset of the neighbors in the conservative step; then, some error must have occurred in the pc, but not necessarily in the fci (since here sepsets can contain nodes outside the neighborhood). If this option is 1 the corresponding triple is marked 'faithful'; if this option is 2, the corresponding triple is marked unfaithful. Second argument: 1 do not consider the initial sepset, 2 also consider the initial sepset.

Value

unfTripl	Vector with triples (coded as number using <code>triple2numb</code>) that were marked as unfaithful.
vers	Vector containing the version (1 or 2) of the corresponding triple saved in <code>unfTripl</code> (1=normal unfaithful triple that is, B is in some sepsets but not all or none; 2=triple coming from <code>version.unf[1]==2</code> that is a and c are indep given the initial sepset but there doesn't exist a subset of the neighbours that d-separates them.)

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>) and Diego Colombo.

See Also

[skeleton](#) for estimating a skeleton using the PC algorithm; [pc](#) for estimating a CPDAG using the PC algorithm; [fci](#) for computing the FCI algorithm.

pcAlgo	<i>PC-Algorithm: Estimate the Underlying Graph (Skeleton) or Equivalence Class (CPDAG) of a DAG</i>
--------	---

Description

This function is DEPRECATED! Use [skeleton](#), [pc](#) or [fci](#) instead.

Usage

```
pcAlgo(dm = NA, C = NA, n=NA, alpha, corMethod = "standard",
        verbose=FALSE, directed=FALSE, G=NULL, datatype = "continuous",
        NAdelete=TRUE, m.max=Inf, u2pd = "rand", psepset=FALSE)
```

Arguments

dm	data matrix; rows correspond to samples, cols correspond to nodes.
C	correlation matrix; this is an alternative for specifying the data matrix.
n	sample size; this is only needed if the data matrix is not provided.
alpha	significance level for the individual partial correlation tests.
corMethod	a character string specifying the method for (partial) correlation estimation. "standard", "QnStable", "Qn" or "ogkQn" for standard and robust (based on the Qn scale estimator without and with OGK) correlation estimation. For robust estimation, we recommend QnStable.
verbose	0-no output, 1-small output, 2-details;using 1 and 2 makes the function very much slower
directed	If FALSE, the underlying skeleton is computed; if TRUE, the underlying CPDAG is computed
G	the adjacency matrix of the graph from which the algorithm should start (logical)
datatype	distinguish between discrete and continuous data
NAdelete	delete edge if pval=NA (for discrete data)
m.max	maximal size of conditioning set
u2pd	Function used for converting skeleton to cpdag. "rand" (use udag2pdag); "relaxed" (use udag2pdagRelaxed); "retry" (use udag2pdagSpecial)
psepset	If true, also possible separation sets are tested.

Value

An object of class "pcAlgo" (see [pcAlgo](#)) containing an undirected graph (object of class "graph", see [graph-class](#) from the package **graph**) (without weights) as estimate of the skeleton or the CPDAG of the underlying DAG.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>) and Martin Maechler.

References

P. Spirtes, C. Glymour and R. Scheines (2000) *Causation, Prediction, and Search*, 2nd edition, The MIT Press.

Kalisch M. and P. B"uhlmann (2007) *Estimating high-dimensional directed acyclic graphs with the PC-algorithm*; JMLR, Vol. 8, 613-636, 2007.

pcAlgo-class	Class "pcAlgo"
--------------	----------------

Description

This class of objects is returned by the functions [skeleton](#) and [pc](#) to represent the (skeleton) of an estimated CPDAG. Objects of this class have methods for the functions [plot](#), [show](#) and [summary](#).

Creation of objects

Objects can be created by calls of the form `new("pcAlgo", ...)` or by function call to [skeleton](#) or [pc](#).

Slots

graph: Object of the class "graph": the undirected or partially directed graph that was estimated.

call: Object of the class "call": the original function call.

n: Object of the class "integer": the sample size used to estimate the graph.

max.ord: Object of class "integer": the maximum size of the conditioning set used in the conditional independence tests of the algorithm.

n.edgetests: Object of class "numeric": the number of conditional independence tests performed by the algorithm.

sepset: Object of class "list": the conditioning sets that led to edge deletions. The set that led to the removal of the edge $i-j$ is saved in either `sepset[[i]][[j]]` or in `sepset[[j]][[i]]`.

pMax: Object of class "matrix": the (i,j) th entry of the matrix contains the maximum p-value of all conditional independence tests for edge $i-j$.

zMin: Deprecated.

Extends

Class "[gAlgo](#)".

Methods

plot signature(x = "pcAlgo"): Plot the resulting graph. If argument "zvalue.lwd" is true, the linewidth an edge reflects zMin, so that thicker lines indicate more reliable dependencies. The argument "lwd.max" controls the maximum linewidth.

show signature(object = "pcAlgo"): Show basic properties of the fitted object

summary signature(object = "pcAlgo"): Show details of the fitted object

Author(s)

Markus Kalisch and Martin Maechler

See Also

[pc](#), [skeleton](#), [fciAlgo](#)

Examples

```
showClass("pcAlgo")

## generate a pcAlgo object
p <- 8
set.seed(45)
myDAG <- randomDAG(p, prob = 0.3)
n <- 10000
d.mat <- rmvDAG(n, myDAG, errDist = "normal")
indepTest <- gaussCItest
suffStat <- list(C = cor(d.mat), n = n)
alpha <- 0.01
pc.fit <- pc(suffStat, indepTest, p, alpha)

## use methods of class pcAlgo
show(pc.fit)
if(require(Rgraphviz))
  plot(pc.fit)
summary(pc.fit)

## access slots of this object
(g <- pc.fit@graph)
str(ss <- pc.fit@sepset, max=1)
```

Description

This function computes partial correlations given a correlation matrix using a recursive algorithm.

Usage

```
pcorOrder(i,j, k, C, cut.at = 0.9999999)
```

Arguments

<code>i, j</code>	integer variable numbers to compute partial correlations of.
<code>k</code>	conditioning set for partial correlations (vector of integers).
<code>C</code>	correlation matrix (matrix)
<code>cut.at</code>	number slightly smaller than one; if c is <code>cut.at</code> , values outside of $[-c, c]$ are set to $-c$ or c respectively.

Details

The partial correlations are computed using a recursive formula if the size of the conditioning set is one. For larger conditioning sets, the pseudoinverse of parts of the correlation matrix is computed (by `pseudoinverse()` from package `corpcor`). The pseudoinverse instead of the inverse is used in order to avoid numerical problems.

Value

The partial correlation of i and j given the set k .

Author(s)

Markus Kalisch <kalisch@stat.math.ethz.ch> and Martin Maechler

See Also

[condIndFisherZ](#) for testing zero partial correlation.

Examples

```
## produce uncorrelated normal random variables
mat <- matrix(rnorm(3*20),20,3)
## compute partial correlation of var1 and var2 given var3
pcorOrder(1,2, 3, cor(mat))

## define graphical model, simulate data and compute
## partial correlation with bigger conditional set
genDAG <- randomDAG(20, prob = 0.2)
dat <- rmvDAG(1000, genDAG)
C <- cor(dat)
pcorOrder(2,5, k = c(3,7,8,14,19), C)
```

pcSelect

*PC-Select: Estimate subgraph around a response variable***Description**

The goal is feature selection: If you have a response variable y and a data matrix dm , we want to know which variables are “strongly influential” on y . The type of influence is the same as in the PC-Algorithm, i.e., y and x (a column of dm) are associated if they are correlated even when conditioning on any subset of the remaining columns in dm . Therefore, only very strong relations will be found and the result is typically a subset of other feature selection techniques. Note that there are also robust correlation methods available which render this method robust.

Usage

```
pcSelect(y, dm, alpha, corMethod = "standard",
         verbose = FALSE, directed = FALSE)
```

Arguments

<code>y</code>	response vector.
<code>dm</code>	data matrix (rows: samples/observations, columns: variables); <code>nrow(dm) == length(y)</code> .
<code>alpha</code>	significance level of individual partial correlation tests.
<code>corMethod</code>	"standard" or "Qn" for standard or robust correlation estimation
<code>verbose</code>	logical or in $\{0, 1, 2\}$; FALSE, 0: no output, TRUE, 1: little output, 2: detailed output. Note that such output makes the function very much slower.
<code>directed</code>	logical; should the output graph be directed?

Details

This function basically applies `pc` on the data matrix obtained by joining y and dm . Since the output is not concerned with the edges found within the columns of dm , the algorithm is adapted accordingly. Therefore, the runtime and the ability to deal with large datasets is typically increased substantially.

Value

<code>G</code>	A boolean vector indicating which column of dm is associated with y
<code>zMin</code>	The minimal z-values when testing partial correlations between y and each column of dm . The larger the number, the more consistent is the edge with the data.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>) and Martin Maechler.

References

Buehlmann, P., Kalisch, M. and Maathuis, M.H. (2009). Variable selection for high-dimensional linear models: partially faithful distributions and the PC-simple algorithm. *To appear in Biometrika*.

See Also

[pc](#) which is the more general version of this function.

Examples

```
p <- 10
## generate and draw random DAG :
set.seed(101)
myDAG <- randomDAG(p, prob = 0.2)
if (require(Rgraphviz)) {
  plot(myDAG, main = "randomDAG(10, prob = 0.2)")
}
## generate 1000 samples of DAG using standard normal error distribution
n <- 1000
d.mat <- rmvDAG(n, myDAG, errDist = "normal")

## let's pretend that the 10th column is the response and the first 9
## columns are explanatory variable. Which of the first 9 variables
## "cause" the tenth variable?
y <- d.mat[,10]
dm <- d.mat[,-10]
(pcS <- pcSelect(d.mat[,10],d.mat[,-10], alpha=0.05))
## You see, that variable 4,5,6 are considered as important
## By inspecting zMin,
with(pcS, zMin[G])
## you can also see that the influence of variable 6
## is very evident from the data (zMin is 21.32, so quite large - as
## a rule of thumb for judging what is large, you could use quantiles
## of the Standard Normal Distribution)

## The result should be the same when using pcAlgo
resU <- pcAlgo(d.mat, alpha = 0.05, corMethod = "standard",directed=TRUE)
resU
if (require(Rgraphviz))
  plot(resU,zvalue.lwd=TRUE)
## as can be seen, the pcAlgo function also finds 4,5,6 as the important
## variables
## Again, variable 6 seems to be very evident from the data
```

pcSelect.presel *Estimate Subgraph around a Response Variable using Preselection*

Description

This function uses [pcSelect](#) to preselect some covariates and then runs [pcSelect](#) again on the reduced data set.

Usage

```
pcSelect.presel(y, dm, alpha, alphapre, corMethod = "standard",
               verbose = 0, directed=FALSE)
```

Arguments

y	response vector.
dm	data matrix (rows: samples, cols: nodes; i.e., <code>length(y) == nrow(dm)</code>).
alpha	significance level of individual partial correlation tests.
alphapre	Significance level for pcSelect in preselection
corMethod	"standard" or "Qn" for standard or robust correlation estimation
verbose	0-no output, 1-small output, 2-details (using 1 and 2 makes the function very much slower)
directed	logical; should the output graph be directed?

Details

First, [pcSelect](#) is run using `alphapre`. Then, only the important variables are kept and [pcSelect](#) is run on them again.

Value

pcs	A boolean vector indicating which column of <code>dm</code> is associated with <code>y</code>
zMin	The minimal z-values when testing partial correlations between <code>y</code> and each column of <code>dm</code> . The larger the number, the more consistent is the edge with the data.
Xnew	Preselected Variables.

Author(s)

Philipp Ruetimann.

See Also

[pcSelect](#)

Examples

```
p <- 10
## generate and draw random DAG :
set.seed(101)
myDAG <- randomDAG(p, prob = 0.2)
if(require(Rgraphviz))
  plot(myDAG, main = "randomDAG(10, prob = 0.2)")

## generate 1000 samples of DAG using standard normal error distribution
n <- 1000
d.mat <- rmvDAG(n, myDAG, errDist = "normal")

## let's pretend that the 10th column is the response and the first 9
## columns are explanatory variable. Which of the first 9 variables
## "cause" the tenth variable?
y <- d.mat[,10]
dm <- d.mat[,-10]
res <- pcSelect.prese1(d.mat[,10],d.mat[,-10],alpha=0.05,alphapre=0.6)
```

pdag2dag

Extend a Partially Directed Acyclic Graph (PDAG) to a DAG

Description

This function extends a PDAG to a DAG, if this is possible.

Usage

```
pdag2dag(g, keepVstruct=TRUE)
```

Arguments

<code>g</code>	input PDAG (graph object)
<code>keepVstruct</code>	If TRUE, the v-structures in <code>g</code> are kept. Otherwise they are ignored and an arbitrary extension is generated.

Details

Direct undirected edges without creating directed cycles or additional v-structures. The PDAG is consistently extended to a DAG using the algorithm by Dor and Tarsi (1992). If no extension is possible, a DAG corresponding to the skeleton of the PDAG is generated and a warning message is produced.

Value

List with two entries: "graph" contains a consistent DAG extension (graph object) and "success" is TRUE iff the extension was possible.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

References

D.Dor, M.Tarsi (1992). A simple algorithm to construct a consistent extension of a partially oriented graph. Technical Report R-185, Cognitive Systems Laboratory, UCLA

Examples

```
p <- 10 # number of random variables
n <- 10000 # number of samples
s <- 0.4 # sparsness of the graph

## generate random data
set.seed(42)
g <- randomDAG(p,s) # generate a random DAG
d <- rmvDAG(n,g) # generate random samples

gSkel <- pcAlgo(d,alpha=0.05) # estimate of the skeleton

gPDAG <- udag2pdag(gSkel)

gDAG <- pdag2dag(gPDAG@graph)
```

pdsep	<i>Compute Possible-D-Sep for Each Node and Adapt Graph Accordingly</i>
-------	---

Description

Computes the Possible-D-Sep for each node and tests conditional independence given all subsets of Possible-D-Sep. The graph is updated accordingly.

Usage

```
pdsep(skel, suffStat, indepTest, p, sepset,
      pMax, NAdelete = TRUE, verbose = FALSE,
      alpha, unfVect = NULL, biCC = FALSE)
```

Arguments

skel	Graph object returned by skeleton .
suffStat	sufficient statistic: A list containing all necessary elements for making conditional independence decisions using function <code>indepTest</code> .

<code>indepTest</code>	predefined function for testing conditional independence. The function is internally called as <code>indepTest(x,y,S,suffStat)</code> for testing conditional independence of x and y given S . Here, x and y are node numbers of the adjacency matrix, S is a (possibly empty) vector of node numbers of the adjacency matrix and <code>suffStat</code> is a list containing all relevant elements for making conditional independence decisions. The return value of <code>indepTest</code> is the p-value of the test for conditional independence.
<code>p</code>	number of variables
<code>sepset</code>	List of length p ; each element of the list contains another list of length p . The element <code>sepset[[x]][[y]]</code> contains the separation set that made the edge between x and y drop out. This object is thought to be obtained from a <code>pcAlgo</code> -object or <code>fciAlgo</code> -object.
<code>pMax</code>	Matrix with the maximal p-values of conditional independence tests in a previous call of <code>skeleton</code> , <code>pc</code> or <code>fci</code> which produced G . This object is thought to be obtained from a <code>pcAlgo</code> -object or <code>fciAlgo</code> -object.
<code>NAdelete</code>	If <code>indepTest</code> returns NA and this option is TRUE, the corresponding edge is deleted. If this option is FALSE, the edge is not deleted.
<code>verbose</code>	If TRUE, detailed output is provided.
<code>alpha</code>	Significance level for the individual conditional independence tests
<code>unfVect</code>	vector containing numbers that encode the unfaithful triple (as returned by <code>pc.cons.intern</code>). This is needed in the conservative FCI.
<code>biCC</code>	logical; if true, only nodes on some path between a and c are considered to be in <code>sepset(a,c)</code> . Uses biconnected components, <code>biConnComp</code> from RBGL .

Details

For a given graph G , a node y is in Possible-D-Sep(x) if $x \neq y$ and there is an undirected path U between x and y in G such that for every subpath a,b,c of U either b is a collider on the subpath, or b is not a definite noncollider on U and a,b and c form a triangle in G .

Each pair of nodes x and y which is connected by an edge is tested for conditional independence given every subset in Possible-D-Sep of x or of y . The conditional independence is tested on significance level α by using the test given in `indepTest`. If the pair of nodes is judged to be independent given set S , then S is recorded in the `sepset` of x,y and in the `sepset` of y,x and the edge is deleted. Otherwise, the edge is not deleted and no entry to the `sepset` is made.

To make the code more efficient, tests that have been done during finding the skeleton are not done again.

Value

A list with several elements:

<code>G</code>	Updated adjacency matrix
<code>sepset</code>	Updated <code>sepset</code>
<code>pMax</code>	Updated matrix containing maximal p-values
<code>allPdsep</code>	Possible d-sep for each node
<code>max.ord</code>	Maximal order of conditioning sets during independence tests
<code>n.edgetests</code>	Number of conditional edgetests performed grouped by size of conditioning set.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

References

P. Spirtes, C. Glymour and R. Scheines (2000). *Causation, Prediction, and Search*, 2nd edition. The MIT Press.

See Also

[qreach](#) for a fast way of finding Possible-D-Sep for a given node; [fci](#) which uses [pdsep](#).

Examples

```
p <- 10
## generate and draw random DAG :
myDAG <- randomDAG(p, prob = 0.2)
## generate 10000 samples of DAG using gaussian distribution
n <- 10000
d.mat <- rmvDAG(n, myDAG, errDist = "normal")

## estimate skeleton
indepTest <- gaussCItest
suffStat <- list(C = cor(d.mat), n = n)
alpha <- 0.01
skel <- skeleton(suffStat, indepTest, p, alpha)

## prepare input for pdsep
sepset <- skel@sepset
pMax <- skel@pMax
n.edgetestsSKEL <- skel@n.edgetests
max.ordSKEL <- skel@max.ord

## call pdsep to find possible d-sep and enhance the skeleton
pdsepRes <- pdsep(skel@graph, suffStat, indepTest, p, sepset, pMax, NAdelete,
verbose = TRUE, alpha)
```

plotAG

Plot partial ancestral graphs (PAG)

Description

This function is DEPRECATED! Use the plot method of the [fciAlgo](#) class instead.

Usage

```
plotAG(amat)
```

Arguments

amat adjacency matrix M with edge marks. The edge marks are coded in the following way: $M[i,j]=M[j,i]=0$: no edge; $M[i,j]=1, M[j,i] \neq 0$: $i \rightarrow j$; $M[i,j]=2, M[j,i] \neq 0$: $i \leftarrow j$; $M[i,j]=3, M[j,i] \neq 0$: $i \leftrightarrow j$.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

See Also

[fci](#)

plotSG

Plot the subgraph around a Specific Node in a Graph Object

Description

Plots a subgraph for a specified starting node and a given graph. The subgraph consists of those nodes that can be reached from the starting node by passing no more than a specified number of edges.

Usage

```
plotSG(graphObj, y, dist, amat = NA, directed = TRUE, main = )
```

Arguments

graphObj an R object of class [graph](#).
y starting node.
dist distance of nodes included in subgraph from starting node y.
amat adjacency matrix of skeleton graph (optional).
directed [logical](#) indicating if the subgraph should be directed.
main title to be used, with a sensible default; see [title](#).

Details

Commencing at the starting point y the function looks for the neighbouring nodes. Beginning with direct parents and children it will continue hierarchically through the distances to y. If directed is true (as per default), the orientation of the edges is taken from the initial graph.

The package **Rgraphviz** must be installed, and is used for the plotting.

Value

the desired subgraph is also plotted and returned (made [invisible](#)).

Author(s)

Daniel Stekhoven (<hoven@stat.math.ethz.ch>)

Examples

```

if (require(Rgraphviz)) {
  ## generate a random DAG:
  p <- 10
  set.seed(45)
  myDAG <- randomDAG(p, prob = 0.3)

  ## plot whole the DAG
  plot(myDAG, main = "randomDAG(10, prob = 0.2)")

  op <- par(mfrow = c(3,2))
  ## plot the neighbours of node number 8 up to distance 1
  plotSG(myDAG, 8, 1, directed = TRUE)
  plotSG(myDAG, 8, 1, directed = FALSE)

  ## plot the neighbours of node number 8 up to distance 2
  plotSG(myDAG, 8, 2, directed = TRUE)
  plotSG(myDAG, 8, 2, directed = FALSE)

  ## plot the neighbours of node number 8 up to distance 3
  plotSG(myDAG, 8, 3, directed = TRUE)
  plotSG(myDAG, 8, 3, directed = FALSE)

  ## Note that the layout of the subgraph might be different than in the
  ## original graph, but the graph structure is identical
  par(op)
}

```

qreach

Compute possible d-sep of node x

Description

This function computes the Possible-D-Sep of node x in a partially directed acyclic graph (PDAG). It is intended for internal use only.

Usage

```
qreach(x, amat, verbose = FALSE)
```

Arguments

x	Column of node in adjacency matrix, of which possible d-sep is to be computed
$amat$	Adjacency matrix as returned by $amat[i,j] = 0$ iff no edge btw i,j $amat[i,j] = 1$ iff $i \rightarrow j$ $amat[i,j] = 2$ iff $i \rightarrow j$
$verbose$	Details on output

Value

Vector of column positions indicating the nodes in Possible-D-Sep of x.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

References

P. Spirtes, C. Glymour and R. Scheines (2000) *Causation, Prediction, and Search*, 2nd edition, The MIT Press.

See Also

[fci](#) and [pdsep](#) which both use this function.

randomDAG

Generate a Directed Acyclic Graph (DAG) randomly

Description

Generate a random Directed Acyclic Graph (DAG). The resulting graph is topologically ordered from low to high node numbers.

Usage

```
randomDAG(n, prob, lB = 0.1, uB = 1)
```

Arguments

n	number of nodes
prob	Probability of connecting a node to another node with higher topological ordering.
lB, uB	Lower and upper limit of weights between connected nodes (chosen uniformly at random).

Details

The n nodes are ordered. Start with first node. Let the number of nodes with higher order be k. Then, the number of neighbouring nodes is drawn as $\text{Bin}(k, \text{prob})$. The neighbours are then drawn without replacement from the nodes with higher order. For each node, a weight is uniformly sampled from lB to uB. This procedure is repeated for the next node in the original ordering and so on.

Value

An object of class "graphNEL", see [graph-class](#) from package **graph**, with n named ("1" to " n ") nodes and directed edges. The graph is topologically ordered. Each edge has a weight between lB and uB .

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>) and Martin Maechler

See Also

[rmvDAG](#) for generating Data according to a DAG; [compareGraphs](#) for comparing the skeleton of a DAG with some other undirected graph (in terms of TPR, FPR and TDR).

Examples

```
if (require(Rgraphviz)) {
  set.seed(101)
  myDAG <- randomDAG(n = 20, prob= 0.2, lB = 0.1, uB = 1)
  plot(myDAG)
}
```

 rfci

Estimate the equivalence class of a MAG (PAG) using the RFCI Algorithm

Description

Estimate the RFCI-PAG from observational data, using the RFCI-algorithm.

Usage

```
rfci(suffStat, indepTest, p, alpha, verbose = FALSE, fixedGaps = NULL,
     fixedEdges = NULL, NAdelete = TRUE, m.max = Inf)
```

Arguments

suffStat	Sufficient statistics: List containing all necessary elements for the conditional independence decisions in the function indepTest.
indepTest	Predefined function for testing conditional independence. The function 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 containing all relevant elements for the conditional independence decisions. The return value of indepTest is the p-value of the test for conditional independence.
p	Number of variables.

alpha	Significance level for the individual conditional independence tests.
verbose	If TRUE, detailed output is provided.
fixedGaps	A logical matrix of dimension $p \times p$. 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 $p \times p$. 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.

Details

This function is rather similar to `fci`. However, it does not compute any possible d-sep sets and thus does not make tests conditioning on them. This makes RFCI much faster than FCI. The orientation rules for v-structures and rule 4 were modified in order to produce a PAG which, in the oracle version, is guaranteed to have the correct ancestral relationships.

Value

An object of `class` `fciAlgo` (see `fciAlgo`) containing the estimated graph (in the form of an adjacency matrix with various possible edge marks), the conditioning sets that lead to edge removals (sepset) and several other parameters.

Author(s)

Diego Colombo and Markus Kalisch (<kalisch@stat.math.ethz.ch>).

References

Diego Colombo, Marloes H. Maathuis, Markus Kalisch, Thomas S. Richardson (2011). Learning high-dimensional DAGs with latent and selection variables. *Arxiv*, <http://arxiv.org/abs/1104.5617>

See Also

`fci` for estimating a PAG using the FCI algorithm; `skeleton` for estimating a skeleton using the PC algorithm; `pc` for estimating a CPDAG using the PC algorithm; `gaussCITest`, `disCITest`, `binCITest` and `dsepTest` as examples for `indepTest`.

Examples

```
#####
## Example without latent variables
#####

set.seed(42)
p <- 7
```

```

## generate and draw random DAG :
myDAG <- randomDAG(p, prob = 0.4)
amat <- wgtMatrix(myDAG)
amat[amat!=0] <- 1
amat <- amat + t(amat)
amat[amat!=0] <- 1

myCPDAG <- dag2cpdag(myDAG)
amat2 <- t(wgtMatrix(myCPDAG))

mycor <- cov2cor(trueCov(myDAG))

## find skeleton and CPDAG using the RFCI algorithm
suffStat <- list(C = cov2cor(trueCov(myDAG)), n = 10^9)
indepTest <- gaussCItest
res <- rfci(suffStat, indepTest, p, alpha = 0.99, verbose=TRUE)

```

rmvDAG

Generate Multivariate Data according to a DAG

Description

Generate multivariate data with dependency structure specified by a (given) DAG (**D**irected **A**cyclic **G**raph) with nodes corresponding to random variables. The DAG has to be **topologically ordered**.

Usage

```

rmvDAG(n, dag,
       errDist = c("normal", "cauchy", "mix", "mixt3", "mixN100",
                  "t4"), mix = 0.1, errMat = NULL)

```

Arguments

dag	a graph object describing the DAG; must contain weights for all the edges. The nodes must be topologically sorted. (For topological sorting use <code>tsort</code> from the RBGL package.)
n	number of samples that should be drawn. (integer)
errDist	Specifies the distribution of each node. Currently, the options "normal", "t4", "cauchy", "mix", "mixt3" and "mixN100" are supported. The first three generate standard normal-, t(df=4)- and cauchy-random numbers. The options containing the word "mix" create standard normal random variables with a mix of outliers. The outliers for the options "mix", "mixt3", "mixN100" are drawn from a standard cauchy, t(df=3) and N(0,100) distribution, respectively. The fraction of cauchy points can be adjusted using the parameter "mix". (string)
mix	For errDist="mix" this parameter specifies the percentage of samples that are drawn from a standard cauchy distribution. (numeric)
errMat	numeric $n * p$ matrix specifying the error vectors e_i (see Details), instead of specifying errDist (and maybe mix).

Details

Each node is visited in the topological order. For each node i we generate a p -dimensional value X_i in the following way: Let X_1, \dots, X_k denote the values of all neighbours of i with lower order. Let w_1, \dots, w_k be the weights of the corresponding edges. Furthermore, generate a random vector e_i according to the specified error distribution. Then, the value of X_i is computed as

$$X_i = w_1 * X_1 + \dots + w_k * X_k + e_i.$$

If node i has no neighbors with lower order, X_i is computed as $X_i = e_i$.

Value

A $n * p$ matrix with the generated data. The p columns correspond to the nodes (i.e., random variables) and each of the n rows correspond to a sample.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>) and Martin Maechler.

See Also

[randomDAG](#) for generating a random DAG; [skeleton](#) and [pc](#) for estimating the skeleton and the CPDAG of a DAG that corresponds to the data.

Examples

```
## generate random DAG
p <- 20
rDAG <- randomDAG(p, prob = 0.2, lB=0.1, uB=1)

if (require(Rgraphviz)) {
  ## plot the DAG
  plot(rDAG, main = "randomDAG(20, prob = 0.2, ..)")
}

## generate 1000 samples of DAG using standard normal error distribution
n <- 1000
d.normMat <- rmvDAG(n, rDAG, errDist="normal")

## generate 1000 samples of DAG using standard t(df=4) error distribution
d.t4Mat <- rmvDAG(n, rDAG, errDist="t4")

## generate 1000 samples of DAG using standard normal with a cauchy
## mixture of 30 percent
d.mixMat <- rmvDAG(n, rDAG, errDist="mix",mix=0.3)

require(MASS) ## for mvrnorm()
Sigma <- toeplitz(ARMAacf(0.2, lag.max = p - 1))
dim(Sigma)# p x p
## *Correlated* normal error matrix "e_i" (against model assumption)
eMat <- mvrnorm(n, mu = rep(0, p), Sigma = Sigma)
d.cnormMat <- rmvDAG(n, rDAG, errMat = eMat)
```

`shd`*Compute Structural Hamming Distance (SHD)*

Description

Computer the Structural Hamming Distance between two graphs. In simple terms, this is the number of edge instertion, deletions or flips in order to transform one graph to another graph.

Usage

```
shd(g1, g2)
```

Arguments

<code>g1</code>	graph object
<code>g2</code>	graph object

Details

The "standard" method invokes a standard correlation estimator. "Qn" invokes a robust, element-wise correlation estimator based on the Qn scale estimate. "QnStable" also uses the Qn scale estimator, but uses an improved way of transforming that into the correlation estimator. "ogkQn" invokes a correlation estimator based on Qn using OGK.

Value

The value of the SHD (numeric).

Author(s)

Markus Kalisch <kalisch@stat.math.ethz.ch> and Martin Maechler

References

I. Tsamardinos, L.E. Brown and C.F. Aliferis (2006). The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm. *JMLR* **65** 31-78.

Examples

```
## generate two graphs
g1 <- randomDAG(10, prob = 0.2)
g2 <- randomDAG(10, prob = 0.2)
## compute SHD
shd.val <- shd(g1, g2)
```

skeleton

*Estimate the skeleton of a DAG using the PC Algorithm***Description**

Estimate the “skeleton” of a directed acyclic graph (DAG) from observational data, using the PC-algorithm.

Usage

```
skeleton(suffStat, indepTest, p, alpha, verbose = FALSE, fixedGaps = NULL,
fixedEdges = NULL, NAdelete = TRUE, m.max = Inf)
```

Arguments

suffStat	Sufficient statistics: List containing all necessary elements for the conditional independence decisions in the function <code>indepTest</code> .
indepTest	Predefined function for testing conditional independence. The function is internally called as <code>indepTest(x, y, S, suffStat)</code> , and tests conditional independence of <code>x</code> and <code>y</code> given <code>S</code> . Here, <code>x</code> and <code>y</code> are variables, and <code>S</code> is a (possibly empty) vector of variables (all variables are denoted by their column numbers in the adjacency matrix). <code>suffStat</code> is a list containing all relevant elements for the conditional independence decisions. The return value of <code>indepTest</code> is the p-value of the test for conditional independence.
p	Number of variables.
alpha	Significance level for the individual conditional independence tests.
verbose	If TRUE, detailed output is provided.
fixedGaps	A logical matrix of dimension $p \times p$. If entry <code>[i, j]</code> or <code>[j, i]</code> (or both) are TRUE, the edge <code>i-j</code> is removed before starting the algorithm. Therefore, this edge is guaranteed to be absent in the resulting graph.
fixedEdges	A logical matrix of dimension $p \times p$. If entry <code>[i, j]</code> or <code>[j, i]</code> (or both) are TRUE, the edge <code>i-j</code> is never considered for removal. Therefore, this edge is guaranteed to be present in the resulting graph.
NAdelete	If <code>indepTest</code> 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.

Details

Under the assumption that the distribution of the observed variables is faithful to a DAG, this function estimates the skeleton of the DAG. The skeleton of a DAG is the undirected graph resulting from removing all arrowheads from the DAG. Edges in the skeleton of a DAG have the following interpretation: there is an edge between `i` and `j` if and only if variables `i` and `j` are conditionally dependent given `S` for all possible subsets `S` of the remaining nodes.

The data are not required to follow a specific distribution, but one should make sure that the conditional independence test used in `indepTest` is appropriate for the data. Pre-programmed versions of `indepTest` are available for Gaussian data (`gaussCItest`), discrete data (`disCItest`), and binary data (see `binCItest`). Users can also specify their own `indepTest` function.

The PC algorithm (Spirtes, Glymour and Scheines, 2000) starts with a complete undirected graph. In each step, it visits all pairs (i, j) of adjacent nodes in the current graph, and determines based on conditional independence tests whether the edge i - j should be removed. In particular, in step m ($m=0,1,\dots$) the algorithm visits all pairs (i, j) of adjacent nodes in the current graph, and the edge between i and j is kept if and only if the null hypothesis " i and j are conditionally independent given S " is rejected at significance level α for all subsets S of size m of the neighbours of i and of the neighbors of j in the current graph (as judged by the function `indepTest`). The algorithm stops when m is larger than the largest neighbourhood size of all nodes, or when m has reached the limit `m.max` defined by the user.

The information in `fixedGaps` and `fixedEdges` is used as follows. The gaps given in `fixedGaps` are introduced in the very beginning of the algorithm by removing the corresponding edges from the complete undirected graph. Pairs (i, j) in `fixedEdges` are skipped in all steps of the algorithm, so that these edges remain in the graph.

Note: Throughout, the algorithm works with the column positions of the variables in the adjacency matrix, and not with the names of the variables.

Value

An object of class "`pcAlgo`" (see `pcAlgo`) containing an estimate of the skeleton of the underlying DAG, the conditioning sets that led to edge removals (`sepset`) and several other parameters.

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>) and Martin Maechler.

References

- P. Spirtes, C. Glymour and R. Scheines (2000) *Causation, Prediction, and Search*, 2nd edition, MIT Press.
- M. Kalisch and P. Buehlmann (2007) *Estimating high-dimensional directed acyclic graphs with the PC-algorithm*, JMLR **8** 613-636.

See Also

`pc` for generating a partially directed graph using the PC algorithm; `udag2pdag` for converting the skeleton to a CPDAG; `gaussCItest`, `disCItest`, `binCItest` and `dsepTest` as examples for `indepTest`.

Examples

```
#####
## Using Gaussian Data
#####
## Load predefined data
data(gmG)
```

```

n <- nrow(gmG$x)
p <- ncol(gmG$x)

## define independence test (partial correlations)
indepTest <- gaussCItest
## define sufficient statistics
suffStat <- list(C = cor(gmG$x), n = n)
## estimate Skeleton
alpha <- 0.01
skeleton.fit <- skeleton(suffStat, indepTest, p, alpha)
if (require(Rgraphviz)) {
## show estimated Skeleton
par(mfrow=c(1,2))
plot(skeleton.fit, main = "Estimated Skeleton")
plot(gmG$g, main = "True DAG")
}
#####
## Using d-separation oracle
#####
## define independence test
indepTest <- dsepTest
## define sufficient statistics (d-separation oracle)
suffStat <- list(g = gmG$g, jp = RBGL::johnson.all.pairs.sp(gmG$g))
## estimate Skeleton
alpha <- 0.01 ## value is irrelevant as dsepTest returns either 0 or 1
fit <- skeleton(suffStat, indepTest, p, alpha)
if (require(Rgraphviz)) {
## show estimated Skeleton
plot(fit, main = "Estimated Skeleton")
plot(gmG$g, main = "True DAG")
}
#####
## Using discrete data
#####
## Load data
data(gmD)
p <- ncol(gmD$x)
## define independence test (G^2 statistics)
indepTest <- disCItest
## define sufficient statistics
suffStat <- list(dm = gmD$x, nlev = c(3,2,3,4,2), adaptDF = FALSE)
## estimate Skeleton
alpha <- 0.01
skeleton.fit <- skeleton(suffStat, indepTest, p, alpha, verbose = TRUE)
if (require(Rgraphviz)) {
## show estimated Skeleton
par(mfrow = c(1,2))
plot(skeleton.fit, main = "Estimated Skeleton")
plot(gmD$g, main = "True DAG")
}

#####
## Using binary data

```

```
#####
## Load binary data
data(gmB)
p <- ncol(gmB$x)
## define independence test
indepTest <- binCItest
## define sufficient statistics
suffStat <- list(dm = gmB$x, adaptDF = FALSE)
## estimate Skeleton
alpha <- 0.01
skeleton.fit <- skeleton(suffStat, indepTest, p, alpha, verbose = TRUE)
if (require(Rgraphviz)) {
## show estimated Skeleton
plot(skeleton.fit, main = "Estimated Skeleton")
plot(gmB$g, main = "True DAG")
}
```

udag2pag

Extend a pcAlgo-object containing a skeleton to a PAG

Description

This function extends a pcAlgo-object containing a skeleton and corresponding conditional independence information to a Partial Ancestral Graph (PAG). The pcAlgo-object must have been estimated with the option `psepset=TRUE`. The result is an adjacency matrix indicating also the edge marks.

Usage

```
udag2pag(gInput, rules, verbose, unfVect = NULL)
```

Arguments

<code>gInput</code>	pcAlgo-object containing skeleton and cond. ind. information; must have been estimated with <code>psepset=TRUE</code>
<code>rules</code>	array of length 10 containing TRUE or FALSE for each rule. TRUE in position <i>i</i> means that rule <i>i</i> (R_i) will be used. Per default, all rules are set to TRUE.
<code>verbose</code>	0: No output; 1: Details. Default is 0.
<code>unfVect</code>	Vector containing numbers that encode the unfaithful triple (as returned by <code>pc.cons.intern</code>). This is needed in the conservative FCI.

Details

The skeleton is extended to a PAG using rules by Zhang (see References). Note that the algorithm works with columns' position of the adjacency matrix and not with the names of the variables.

Value

The output is an adjacency matrix M with edge marks. The edge marks are coded in the following way: $M[i,j]=M[j,i]=0$: no edge; $M[i,j]=1, M[j,i] \neq 0$: $i \rightarrow j$; $M[i,j]=2, M[j,i]=0$: $i \rightarrow j$; $M[i,j]=3, M[j,i]=0$: $i \rightarrow j$.

Author(s)

Diego Colombo (<colombo@stat.math.ethz.ch>)

References

J. Zhang (2008). On the completeness of orientation rules for causal discovery in the presence of latent confounders and selection bias. *Artificial Intelligence* **172** 1873-1896.

See Also

[pc](#)

Examples

```
## generate and draw random DAG
## this example is taken from Zhang (2008), Fig. 6, p.1882 (see references)
amat <- t(matrix(c(0,1,0,0,1, 0,0,1,0,0, 0,0,0,1,0, 0,0,0,0,0, 0,0,0,1,0),5,5))
colnames(amat) <- rownames(amat) <- as.character(1:5)
V <- as.character(1:5)
edL <- vector("list",length=5)
names(edL) <- V
edL[[1]] <- list(edges=c(2,4),weights=c(1,1))
edL[[2]] <- list(edges=3,weights=c(1))
edL[[3]] <- list(edges=5,weights=c(1))
edL[[4]] <- list(edges=5,weights=c(1))
g <- new("graphNEL", nodes=V, edgeL=edL,edgemode="directed")
if (require(Rgraphviz)) {
  plot(g)
}

## define the latent variable
L <- 1

## generate 100000 samples of DAG using standard normal error distribution
n <- 100000
d.mat <- rmvDAG(n, g, errDist = "normal")

## delete rows and columns corresponding to the latent variable
d.mat <- d.mat[-L,-L]

## estimate the skeleton of given data using psepset=TRUE
(resD <- pcAlgo(d.mat, alpha=0.05, psepset=TRUE))

## extend the pcalgo-object into a PAG using all 10 rules
rules <- rep(TRUE,10)
```

```

resP <- udag2pdag(resD, rules=rules, verbose=1)
colnames(resP) <- rownames(resP) <- graph::nodes(g)[-L]

if (require(Rgraphviz)) {
  ## plot the original DAG and the PAG
  op <- par(mfrow=c(1,2))
  plot(g, main="original (random) DAG")
  plotAG(resP)
  par(op)
}

```

udag2pdag

Extend a pcAlgo-object containing a skeleton to a PDAG

Description

This function extends an object of class "[pcAlgo](#)" containing a skeleton and corresponding conditional independence information to a Partially Directed Acyclic Graph (PDAG). The result is a "pcAlgo"-object as well.

Usage

```
udag2pdag(gInput, verbose)
```

Arguments

gInput	pcAlgo-object containing skeleton and cond. ind. information
verbose	0: No output; 1: Details

Details

The skeleton is extended to a PDAG using rules by Spirtes and Pearl (see References).

Value

pcObj	Oriented pc-Object
-------	--------------------

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

References

P. Spirtes, C. Glymour and R. Scheines (2000) *Causation, Prediction, and Search*, 2nd edition, The MIT Press.

J. Pearl (2000), *Causality*, Cambridge University Press.

See Also

[pdag2dag](#), [dag2cpdag](#), [udag2pdag](#), [udag2pdagRelaxed](#), [udag2pdagSpecial](#)

Examples

```
## simulate data
set.seed(123)
p <- 10
myDAG <- randomDAG(p, prob = 0.2)
trueCPDAG <- dag2cpdag(myDAG)
n <- 1000
d.mat <- rmvDAG(n, myDAG, errDist = "normal")

## define independence test (partial correlations)
indepTest <- gaussCitest
## define sufficient statistics
suffStat <- list(C = cor(d.mat), n = n)
## estimate skeleton
resU <- skeleton(suffStat, indepTest, p, 0.05)

## orient edges using three different methods
resD1 <- udag2pdagRelaxed(resU, verbose=0)
resD2 <- udag2pdagSpecial(resU, verbose=0, n.max=100)
resD3 <- udag2pdag(resU, verbose=0)
```

udag2pdagRelaxed

Extend a pcAlgo-object containing a skeleton to a PDAG

Description

This function extends a pcAlgo-object containing a skeleton and corresponding conditional independence information to a Partially Directed Acyclic Graph (PDAG). The result is a pcAlgo-object as well. There is no check whether the result is extendible to a DAG

Usage

```
udag2pdagRelaxed(gInput, verbose, unfaithful=vector())
```

Arguments

gInput	pcAlgo-object containing skeleton and cond. ind. information
verbose	0: No output; 1: Details
unfaithful	Vector containing numbers that encode the unfaithful triple (as returned by pc.cons.intern). This is needed in the conservative PC.

Details

The skeleton is extended to a PDAG using rules by Spirtes and Pearl (see References). There is no test whether the result is really extendible.

Value

pcObj Oriented pc-Object

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

References

P. Spirtes, C. Glymour and R. Scheines (2000) *Causation, Prediction, and Search*, 2nd edition, The MIT Press.

J. Pearl (2000), *Causality*, Cambridge University Press.

See Also

[pdag2dag](#), [dag2cpdag](#), [udag2pdag](#), [udag2pdagRelaxed](#), [udag2pdagSpecial](#)

Examples

```
## simulate data
set.seed(123)
p <- 10
myDAG <- randomDAG(p, prob = 0.2)
trueCPDAG <- dag2cpdag(myDAG)
n <- 1000
d.mat <- rmvDAG(n, myDAG, errDist = "normal")

## define independence test (partial correlations)
indepTest <- gaussCitest
## define sufficient statistics
suffStat <- list(C = cor(d.mat), n = n)
## estimate skeleton
resU <- skeleton(suffStat, indepTest, p, 0.05)

## orient edges using three different methods
resD1 <- udag2pdagRelaxed(resU, verbose=0)
resD2 <- udag2pdagSpecial(resU, verbose=0, n.max=100)
resD3 <- udag2pdag(resU, verbose=0)
```

udag2pdagSpecial	<i>Extend a pcAlgo-object containing a skeleton to a PDAG using different methods if problems occur</i>
------------------	---

Description

This function extends a pcAlgo-object containing a skeleton and corresponding conditional independence information to a Partially Directed Acyclic Graph (PDAG). The result is a pcAlgo-object as well.

Usage

```
udag2pdagSpecial(gInput, verbose, n.max=100)
```

Arguments

gInput	pcAlgo-object containing skeleton and cond. ind. information
verbose	0: No output; 1: Details
n.max	Maximum number of tries for reorienting doubly visited edges.

Details

The skeleton is extended to a PDAG using rules by Spirtes and Pearl (see References). If, after orienting the v-structures, the graph is not extendible to a DAG, the following mechanisms try to solve the problem: There might be edges, that were oriented twice while forming colliders. In this case, try all possible combinations of reorientations and check whether any reorientation is extendible. Take the first one that is extendible and make no more than n.max tries. If this fails, the original graph is extended arbitrarily to a DAG that fits on the skeleton. v-structures might have changed. The resulting DAG is then transformed to its CPDAG.

Value

pcObj	Oriented pc-Object
evisit	Matrix counting the number of orientation attempts per edge
xtbl.orig	Is original graph with v-structure extendible
xtbl	Is final graph with v-structure extendible
amat0	Adj.matrix of original graph with v-structures
amat1	Adj.matrix of graph with v-structures after reorienting edges from double edge visits
status	0: original try is extendible; 1: reorienting double edge visits helps; 2: orig. try is not extendible; reorienting double visits don't help; result is acyclic, has orig. v-structures, but perhaps additional v-structures
counter	Number of reorientation tries until success or max.tries

Author(s)

Markus Kalisch (<kalisch@stat.math.ethz.ch>)

References

P. Spirtes, C. Glymour and R. Scheines (2000) *Causation, Prediction, and Search*, 2nd edition, The MIT Press.

J. Pearl (2000), *Causality*, Cambridge University Press.

See Also

[pdag2dag](#), [dag2cpdag](#), [udag2pdag](#), [udag2pdagRelaxed](#)

Examples

```
## simulate data
set.seed(123)
p <- 10
myDAG <- randomDAG(p, prob = 0.2)
trueCPDAG <- dag2cpdag(myDAG)
n <- 1000
d.mat <- rmvDAG(n, myDAG, errDist = "normal")

## define independence test (partial correlations)
indepTest <- gaussCItest
## define sufficient statistics
suffStat <- list(C = cor(d.mat), n = n)
## estimate skeleton
resU <- skeleton(suffStat, indepTest, p, 0.05)

## orient edges using three different methods
resD1 <- udag2pdagRelaxed(resU, verbose=0)
resD2 <- udag2pdagSpecial(resU, verbose=0, n.max=100)
resD3 <- udag2pdag(resU, verbose=0)
```

Index

- *Topic **arith**
 - getNextSet, 21
- *Topic **classes**
 - fciAlgo-class, 18
 - pcAlgo-class, 43
- *Topic **datagen**
 - randomDAG, 55
 - rmvDAG, 58
- *Topic **datasets**
 - gmB, 23
 - gmD, 23
 - gmG, 24
 - gmI, 25
 - gmL, 26
- *Topic **graphs**
 - beta.special, 3
 - beta.special.pcObj, 4
 - compareGraphs, 5
 - corGraph, 9
 - dag2cpdag, 10
 - fci, 14
 - ida, 30
 - idaFast, 33
 - pc, 36
 - pc.cons.intern, 40
 - pcAlgo, 42
 - pcSelect, 46
 - pcSelect.presel, 48
 - pdag2dag, 49
 - plotAG, 52
 - plotSG, 53
 - randomDAG, 55
 - rfci, 56
 - shd, 60
 - skeleton, 61
 - udag2pag, 64
 - udag2pdag, 66
 - udag2pdagRelaxed, 67
 - udag2pdagSpecial, 69
- *Topic **htest**
 - condIndFisherZ, 7
- *Topic **models**
 - beta.special, 3
 - beta.special.pcObj, 4
 - corGraph, 9
 - dag2cpdag, 10
 - fci, 14
 - ida, 30
 - idaFast, 33
 - pc, 36
 - pc.cons.intern, 40
 - pcAlgo, 42
 - pcSelect, 46
 - pcSelect.presel, 48
 - plotAG, 52
 - rfci, 56
 - skeleton, 61
 - udag2pag, 64
 - udag2pdag, 66
 - udag2pdagRelaxed, 67
 - udag2pdagSpecial, 69
- *Topic **multivariate**
 - beta.special, 3
 - beta.special.pcObj, 4
 - condIndFisherZ, 7
 - dag2cpdag, 10
 - fci, 14
 - ida, 30
 - idaFast, 33
 - mcor, 35
 - pc, 36
 - pc.cons.intern, 40
 - pcAlgo, 42
 - pcorOrder, 44
 - pcSelect, 46
 - pcSelect.presel, 48
 - plotAG, 52
 - rfci, 56

- rmvDAG, 58
- skeleton, 61
- udag2pag, 64
- udag2pdag, 66
- udag2pdagRelaxed, 67
- udag2pdagSpecial, 69
- *Topic **robust**
 - mcor, 35
- *Topic **utilities**
 - getNextSet, 21
- allDags, 31
- beta.special, 3
- beta.special.pcObj, 4, 4
- biConnComp, 51
- binCITest, 4, 11, 14, 17, 21, 28, 39, 57, 62
- causalEffect (ida), 30
- class, 12, 13, 16, 38, 42, 57, 62
- combn, 22
- compareGraphs, 5, 56
- condIndFisherZ, 7, 45
- corGraph, 9
- covOGK, 35
- dag2cpdag, 4, 10, 67, 68, 70
- disCITest, 5, 11, 14, 17, 21, 30, 39, 57, 62
- dsep, 12, 14
- dsepTest, 5, 11, 13, 13, 17, 21, 39, 57, 62
- fci, 4, 11, 13, 14, 19, 20, 28, 30, 40–42, 51–53, 55, 57
- fciAlgo, 16, 44, 52, 57
- fciAlgo-class, 18
- gAlgo, 19, 43
- gaussCITest, 5, 11, 14, 17, 20, 39, 57, 62
- getNextSet, 21
- gmB, 23
- gmD, 23
- gmG, 24
- gmI, 25
- gmL, 26
- graph, 53
- gSquareBin, 5, 27, 29, 30
- gSquareDis, 11, 28, 29
- ida, 3, 4, 30, 33, 34
- idaFast, 4, 32, 33, 34
- identical, 25, 26
- invisible, 53
- johnson.all.pairs.sp, 12, 13
- logical, 8, 53
- mcor, 35
- pc, 4, 11, 13, 16, 17, 20, 28, 30–34, 36, 40–44, 46, 47, 51, 57, 59, 62, 65
- pc.cons.intern, 40, 51
- pcAlgo, 4, 20, 38, 42, 42, 62, 66
- pcAlgo-class, 43
- pcorOrder, 8, 35, 44
- pcSelect, 46, 48
- pcSelect.presel, 48
- pdag2dag, 10, 49, 67, 68, 70
- pdsep, 16, 17, 50, 52, 55
- plot, 19
- plot, fciAlgo-method (fciAlgo-class), 18
- plot, pcAlgo-method (pcAlgo-class), 43
- plotAG, 52
- plotSG, 53
- pseudoinverse, 45
- Qn, 35
- qnorm, 7
- qreach, 17, 52, 54
- randomDAG, 6, 55, 59
- rfci, 56
- rmvDAG, 56, 58
- shd, 60
- show, 19
- show, fciAlgo-method (fciAlgo-class), 18
- show, pcAlgo-method (pcAlgo-class), 43
- skeleton, 4, 11, 13, 16, 17, 19, 20, 22, 28, 30, 37, 39, 41–44, 50, 51, 57, 59, 61
- summary, 19
- summary, fciAlgo-method (fciAlgo-class), 18
- summary, pcAlgo-method (pcAlgo-class), 43
- title, 53
- tsort, 58
- udag2pag, 64
- udag2pdag, 10, 39, 62, 66, 67, 68, 70

udag2pdagRelaxed, [67](#), [67](#), [68](#), [70](#)

udag2pdagSpecial, [67](#), [68](#), [69](#)

zStat, [20](#)

zStat (condIndFisherZ), [7](#)