

# Package ‘pcalg’

September 23, 2009

**Version** 0.1-9

**Date** 2009-3-03

**Author** Markus Kalisch, Martin Maechler, Diego Colombo

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

**Title** Estimation of CPDAG/PAG and causal inference

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

**Depends** methods, MASS, graph, robustbase, Rgraphviz, ggm, mnormt, vcd, RBGL, sfsmisc

**License** GPL (>= 2)

**Repository** CRAN

**Date/Publication** 2009-09-23 16:45:42

## R topics documented:

beta.special . . . . .	2
beta.special.pcObj . . . . .	4
compareGraphs . . . . .	6
condIndFisherZ . . . . .	7
corGraph . . . . .	9
dag2cpdag . . . . .	10
decHeur . . . . .	11
getNextSet . . . . .	13
mcor . . . . .	14
pcAlgo . . . . .	15
pcAlgo-class . . . . .	18

pcorOrder . . . . .	19
pcSelect . . . . .	20
pcSelect.presel . . . . .	22
pdag2dag . . . . .	23
plotAG . . . . .	24
randomDAG . . . . .	26
rmvDAG . . . . .	27
shd . . . . .	29
udag2pag . . . . .	30
udag2pdag . . . . .	31
udag2pdagRelaxed . . . . .	33
udag2pdagSpecial . . . . .	34

<b>Index</b>	<b>36</b>
--------------	-----------

---

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

---

## Description

This function computes a set of intervention effects of one variable onto another variable.

## Usage

```
beta.special (dat=NA, x.pos, y.pos, verbose=0, a=0.01, myDAG=NA, myplot=FALSE, perfect=FALSE)
```

## 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

**Details**

After estimating a CPDAG, intervention effects of x on y are computed. For details, see references.

**Value**

estimates of intervention effects

**Author(s)**

Markus Kalisch ([kalisch@stat.math.ethz.ch](mailto:kalisch@stat.math.ethz.ch))

**References**

M.H. Maathuis, M. Kalisch, P. Bühlmann (2009), *Estimating high-dimensional intervention effects from observational data*; Annals of Statistics, 2009.

**See Also**

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

**Examples**

```
#####
## Global
#####
set.seed(125)
p <- 10
n <- 100000

myDAG <- randomDAG(p, prob = 0.4)
plot(dag2cpdag(myDAG))
cmat <- cov2cor(trueCov(myDAG))
d.mat <- rmvDAG(n, myDAG, errDist = "normal")

x.pos <- 1
y.pos <- 5
## true value
true.eff <- causalEffect(myDAG, y.pos, x.pos)
## value using oracle (true value should be element of this)
est.eff.p <- beta.special(x.pos=x.pos, y.pos=y.pos, myDAG=myDAG, perfect=TRUE, method="global")
## value using data (true value should come close to some element of this)
est.eff.e <- beta.special(dat=d.mat, x.pos=x.pos, y.pos=y.pos, perfect=FALSE, method="global", my

true.eff
est.eff.p
est.eff.e

## Setting perfect&global always contains exact value of true value
## OK

#####
```

```

## Local
#####
set.seed(125)
p <- 10
n <- 100000

myDAG <- randomDAG(p, prob = 0.4)
cmat <- cov2cor(trueCov(myDAG))
d.mat <- rmvDAG(n, myDAG, errDist = "normal")

x.pos <- 1
y.pos <- 5
## true value
true.eff <- causalEffect(myDAG,y.pos,x.pos)
## value using oracle (true value should be element of this)
est.eff.p <- beta.special(x.pos=x.pos,y.pos=y.pos,myDAG=myDAG,perfect=TRUE,method="local")
## value using data (true value should come close to some element of this)
est.eff.e <- beta.special(dat=d.mat,x.pos=x.pos,y.pos=y.pos,perfect=FALSE,method="local")
plot(pcAlgo(d.mat,alpha=0.01,directed=TRUE))

true.eff
est.eff.p
est.eff.e

```

---

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

---

## Description

This function computes a set of intervention effects of one variable onto another variable in a fast way. The pc-object has to be precomputed.

## Usage

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

## Arguments

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

**Details**

Estimate the intervention effect of  $x$  on  $y$ ; the pcObj has to be precomputed. This method is intended to be a fast version of

```
beta.special(dat=NA,x.pos,y.pos,verbose=0,a=NA,myDAG=NA,myplot=FALSE,perfect=FALSE,method="local",collTest=scaled.data=FALSE,u2pd="relaxed")
```

Thus, this is a faster version for the local method given a precomputed PC-Algo Object (relaxed udag2pdag, so CPDAG might not be a real CPDAG; this does not matter, since we try not to extend).

If this function is used repeatedly on the same pc-object (e.g. for different values of  $x$  and  $y$ ), the matrices amat, amatSkel and t.amat can be precomputed in order to be more efficient (see the first entry in the code for further details).

**Value**

estimates of intervention effects

**Author(s)**

Markus Kalisch ([kalisch@stat.math.ethz.ch](mailto:kalisch@stat.math.ethz.ch))

**References**

M.H. Maathuis, M. Kalisch, P. Bühlmann (2009), *Estimating high-dimensional intervention effects from observational data*; Annals of Statistics, 2009.

**See Also**

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

**Examples**

```
set.seed(125)
p <- 10
n <- 10000

myDAG <- randomDAG(p, prob = 0.4)

cov.t <- trueCov(myDAG)
pcObj <- pcAlgo.Perfect(cov2cor(cov.t), directed=TRUE)
d.mat <- rmvDAG(n, myDAG, errDist = "normal")
cov.e <- cov(d.mat)
pcObj.est <- pcAlgo(d.mat, alpha=0.01, directed=TRUE)

x.pos <- 1
y.pos <- 5
true.eff <- causalEffect(myDAG, y.pos, x.pos)
est.eff.p <- beta.special.pcObj(x.pos=x.pos, y.pos=y.pos, pcObj=pcObj, mcov=cov.t)
est.eff.e <- beta.special.pcObj(x.pos=x.pos, y.pos=y.pos, pcObj=pcObj.est, mcov=cov.e)

true.eff
est.eff.p
```

```
est.eff.e
```

---

```
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

<code>gl</code>	estimated graph (graph object)
<code>gt</code>	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

<code>tpr</code>	True Positive Rate: Number of correctly found edges (in estimated graph) divided by number of true edges (in true graph)
<code>fpr</code>	False Positive Rate: Number of incorrectly found edges divided by number of true gaps
<code>tdr</code>	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](mailto: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 <- addEdge("A","C", gt,1)

## compare the two graphs
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**

<code>x, y, S</code>	it is tested, whether <code>x</code> and <code>y</code> are conditionally independent given the subset <code>S</code> of the remaining nodes. <code>x</code> , <code>y</code> , <code>S</code> all are integers, corresponding to variable or node numbers.
<code>C</code>	correlation matrix of nodes
<code>n</code>	integer specifying the number of observations (“samples”) used to estimate the correlation matrix <code>C</code> .
<code>cutoff</code>	numeric cutoff for significance level of individual partial correlation tests. Must be set to <code>qnorm(1 - alpha/2)</code> for a test significance level of <code>alpha</code> .
<code>verbose</code>	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](mailto:kalisch@stat.math.ethz.ch)) and Martin Maechler

## References

Markus Kalisch and Peter Bühlmann (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](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 on a given significance level  $\alpha$  (*alpha*).

### Usage

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

### Arguments

dm	numeric matrix with rows as samples and columns as variables.
alpha	significance level for correlation test (numeric)
Cmethod	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](mailto: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)

## ``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). We use the algorithm used by Chickering (see References).

### Value

An adjacency matrix containing the CPDAG.

### Author(s)

Markus Kalisch ([kalisch@stat.math.ethz.ch](mailto:kalisch@stat.math.ethz.ch))

### References

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

### See Also

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

**Examples**

```

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

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

res <- dag2cpdag(g)

```

---

decHeur

---

*Decision Heuristic: Should robust method for PC-algorithm be used?*


---

**Description**

Simple Heuristic for deciding whether robust PC-algorithm should be used.

**Usage**

```
decHeur(dat, gam=0.05, sim.method="t", est.method="o", n.sim=100, two.sided=FALSE, verbose)
```

**Arguments**

dat	Data matrix (cols=variables, rows=samples)
gam	Significance level for test
sim.method	Reference distribution; "n" for Normal, "t" for N+10% t3
est.method	Estimation method of correlation matrix; "s" for standard, "o" for OGK using Qn (robust)
n.sim	Number of samples drawn from reference distribution
two.sided	Should a two-sided test be used?
verbose	Run in verbose mode

**Details**

Simulation studies show that the standard PC-algorithm already is rather insensitive to outliers, provided, they are not too severe. The effect of very heavy outliers can be dramatically reduced by using the robust PC-algorithm; this increases the computational burden by roughly one order of magnitude.

We provide a simple method for deciding whether data at hand has worse outliers than a given reference distribution. Using this, we see two heuristics for deciding whether to use the robust version of the PC-algorithm or not. On the one hand, one could use the normal distribution as reference distribution and apply the robust PC-algorithm to all data that seem to contain more outliers than an appropriate normal distribution (Heuristic A). On the other hand, one could, inspired by the results of simulation studies, only want to apply the robust method in the case where the contamination is worse than a normal distribution with 10% outliers from a  $t_3$  distribution. Then, we would use

a normal distribution with 10% outliers from a  $t_3$  distribution as reference distribution (Heuristic B).

In order to decide whether data has worse outliers than a given reference distribution, we proceed as follows. We compute a robust estimate of the covariance matrix of the data (e.g. OGK with Qn-estimator) and simulate (several times) data from the reference distribution with this covariance matrix. For each dimension  $i$  ( $1 \leq i \leq p$ ), we compute the ratio of standard deviation  $\sigma_i$  and a robust version of it  $s_i$  (e.g., Qn-estimator) and compute the average over all dimensions. (Since the main input for the PC-algorithm are correlation estimates which can be expressed in terms of scale estimates, we base our test statistics on scale estimates.) Thus, we obtain the distribution of this averaged ratio  $R = \frac{1}{p} \sum_{i=1}^p \sigma_i / s_i$  under the null hypothesis that the data can be explained by the reference distribution with given covariance matrix. We now can test this null hypothesis by using the ratio computed with the current data set  $r = \frac{1}{p} \sum_{i=1}^p \hat{\sigma}_i / \hat{s}_i$  on a given significance level.

### Value

tvec	Simulated values of test statistic
tval	Observed value of test statistic
outlier	Is robust method suggested? (TRUE=Suggested)

### Author(s)

Markus Kalisch ([kalisch@stat.math.ethz.ch](mailto:kalisch@stat.math.ethz.ch))

### See Also

[pcAlgo](#) which can be used with standard and robust correlation estimate.

### Examples

```
set.seed(123)
## generate a data set for the example
p <- 5
myDAG <- randomDAG(p, prob = 0.6)
n <- 1000
## data without outlier
datN <- rmvDAG(n, myDAG, errDist = "normal")
## data with severe outlier (10% Cauchy)
datC <- rmvDAG(n, myDAG, errDist = "mix")
n.sim <- 20
gam <- 0.05
sim.method <- "t"
est.method <- "o"
decHeur(datN, gam, sim.method, est.method, n.sim=n.sim, two.sided=FALSE, verbose=TRUE)
decHeur(datC, gam, sim.method, est.method, n.sim=n.sim, two.sided=FALSE, verbose=TRUE)
```

---

getNextSet	<i>Iteration through a list of all combinations of choose(n,k)</i>
------------	--

---

### 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:

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

### Author(s)

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

### 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

```

---

mcor

---

*Compute 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 <a href="#">Qn</a> ), robust (Qn using OGK, see <a href="#">covOGK</a> ) 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 according to the specified method.

### Author(s)

Markus Kalisch ([kalisch@stat.math.ethz.ch](mailto: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 and `condIndFisherZ` for testing zero partial correlation.

## 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 a lot correlation
mcor(cbind(x,yNoise), method="ogkQn")  ## dito
```

---

pcAlgo

*PC-Algorithm: Estimate the Underlying Graph (Skeleton) or Equivalence Class (CPDAG) of a DAG*

---

## Description

Estimate the underlying graph (`ugraph` or “skeleton” (structure) of a DAG (**D**irected **A**cyclic **G**raph) from data using the PC-algorithm. Alternatively, one can directly infer the equivalence class of the underlying structure, which is uniquely represented by its CPDAG (**C**ompleted **P**artially **D**irected **A**cyclic **G**raph). For continuous or discrete data.

## 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

<code>dm</code>	data matrix; rows correspond to samples, cols correspond to nodes.
<code>C</code>	correlation matrix; this is an alternative for specifying the data matrix.
<code>n</code>	sample size; this is only needed if the data matrix is not provided.
<code>alpha</code>	significance level for the individual partial correlation tests.

<code>corMethod</code>	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.
<code>verbose</code>	0-no output, 1-small output, 2-details;using 1 and 2 makes the function very much slower
<code>directed</code>	If FALSE, the underlying skeleton is computed; if TRUE, the underlying CPDAG is computed
<code>G</code>	the adjacency matrix of the graph from which the algorithm should start (logical)
<code>datatype</code>	distinguish between discrete and continuous data
<code>NAdelete</code>	delete edge if pval=NA (for discrete data)
<code>m.max</code>	maximal size of conditioning set
<code>u2pd</code>	Function used for converting skeleton to cpdag. "rand" (use <code>udag2pdag</code> ); "relaxed" (use <code>udag2pdagRelaxed</code> ); "retry" (use <code>udag2pdagSpecial</code> )
<code>pseptest</code>	If true, also possible separation sets are tested.

## Details

The algorithm starts with a complete undirected graph or with `G` (if given). The following text explains the continuous case. For the discrete case, replace (partial) correlation tests by (conditional) independence tests. In a first sweep, an edge  $ij$  is kept only if  $H_0 : Cor(X_i, X_j) = 0$  can be rejected on significance level `alpha`. All ordered pairs  $ij$  of nodes of the resulting graph are then swept again. An edge  $ij$  is kept only if  $H_0 : Cor(X_i, X_j | X_k) = 0$  can be rejected for all neighbours  $k$  of  $i$  in the current graph. Again, the remaining edges are swept. This time, an ordered pair (edge)  $ij$  is kept only if  $H_0 : Cor(X_i, X_j | X_a, X_b) = 0$  can be rejected for all subsets of size two  $(a, b)$  of the neighbours of  $i$  in the remaining graph. In the next step, the remaining edges are tested using all subsets of size three, then of size four and so on. The algorithm stops when the largest neighbourhood is smaller than the size of the conditioning sets.

The partial correlations are computed recursively or via matrix inversion from the correlation matrix, which are computed by the specified method (`corMethod`). The partial correlation tests are based on Fisher's z-transformation. For more details on the methods for computing the correlations see `mcor`.

Note that the algorithm works with columns' position of the adjacency matrix and not with the names of the variables.

## 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. Bühlmann (2007) *Estimating high-dimensional directed acyclic graphs with the PC-algorithm*; JMLR, Vol. 8, 613-636, 2007.

## See Also

[randomDAG](#) for generating a random DAG; [rmvDAG](#) for generating data according to a DAG; [compareGraphs](#) for comparing undirected graphs in terms of TPR, FPR and TDR. Further, [randomGraph](#) (in package **graph**) for other random graph models. [udag2pdag](#) for converting the skeleton to a CPDAG.

## Examples

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

## estimate skeleton and CPDAG of given data
resU <- pcAlgo(d.mat, alpha = 0.05, corMethod = "standard")
resU
plot(resU, zvalue.lwd=TRUE) # << using the plot() method for 'pcAlgo' objects!
str(resU, max = 2)
(c.g <- compareGraphs(myDAG, resU@graph))
## CPDAG
resD <- pcAlgo(d.mat, alpha = 0.05, corMethod =
"standard", directed=TRUE)
plot(resD, zvalue.lwd=TRUE)

## plot the original DAG, the estimated skeleton and the estimated CPDAG:
op <- par(mfrow=c(3,1))
plot(myDAG, main = "original (random)DAG")
plot(resU@graph,
      main = "estimated skeleton from pcAlgo(<simulated, n =
1000>)")
plot(resD@graph, main="estimated CPDAG from pcAlgo(<simulated, n =
1000>)")
par(op)

## generate data containing severe outliers
d.mixmat <- rmvDAG(n, myDAG, errDist = "mix", mix=0.3)
## Compute "classical" and robust estimate of skeleton :
pcC <- pcAlgo(d.mixmat, alpha = 0.01, corMeth = "standard")
pcR <- pcAlgo(d.mixmat, alpha = 0.01, corMeth = "Qn")
```

```

str(pcR, max = 2)
(c.Cg <- compareGraphs(myDAG, pcC@graph))
(c.Rg <- compareGraphs(myDAG, pcR@graph))#-> (.201 0 1) much better
op <- par(mfrow=c(3,1))
  plot(myDAG, main = "original (random) DAG")
  plot(pcC)
  plot(pcR, zvalue.lwd=TRUE, lwd.max=7)
par(op)

```

---

pcAlgo-class

Class "pcAlgo"

---

## Description

The object contains the results of a call of the function [pcAlgo](#).

## Objects from the Class

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

## Slots

**graph:** Object of class "graph" which contains the undirected graph that was estimated  
**call:** Object of class "call" which contains the original call  
**n:** Object of class "integer"; number of samples that were used to estimate the graph  
**max.ord:** Object of class "integer"; maximum size of neighbourhood set that was conditioned on during the algorithm  
**n.edgetests:** Object of class "numeric"; number of edge tests that were performed  
**sepset:** Object of class "list"; the conditioning sets that made an edge drop out during the run of the algorithm  
**zMin:** Object of class "matrix"; contains at entry ij the minimum z-Value of all conditional independence tests of edge ij

## Methods

**plot** signature(x = "pcAlgo"): Plot the resulting undirected graph; if argument "zvalue.lwd" is true, the linewidth of the edges reflects the minimum absolute z-Value that was computed during all tests for the particular edge, i.e., thicker lines show more reliable dependencies; the argument "lwd.max" controls the maximal linewidth.  
**show** signature(object = "pcAlgo"): Show properties of the fitted object  
**summary** signature(object = "pcAlgo"): Show details of the fitted object

## Author(s)

Markus Kalisch and Martin Maechler

**See Also**[pcAlgo](#)

---

`pcorOrder`*Compute Partial Correlations*

---

**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 <code>c</code> 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. To avoid numeric problems in recursions, the partial correlation is set to `cut.at` (= 0.9999999 by default; keep this value, unless you know better!), if it is above (e.g., if 1) and to  $-\text{cut.at}$  if it is below (e.g., if -1).

**Value**

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

**Author(s)**

Markus Kalisch ([kalisch@stat.math.ethz.ch](mailto: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

This function is intended for feature selection: If you have a response variable  $y$  and a data matrix  $dm$ , which columns 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 = 0, directed=FALSE)
```

## Arguments

<code>y</code>	Response Vector (length(y)=nrow(dm))
<code>dm</code>	Data matrix (rows: samples, cols: nodes)
<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>	0-no output, 1-small output, 2-details (using 1 and 2 makes the function very much slower)
<code>directed</code>	Boolean; should the output graph be directed?

## Details

This function basically applies `pcAlgo` 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 quite a lot.

**Value**

G A boolean vector indicating which column of `dm` is associated with `y`

`zMin` 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](mailto: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.

**See Also**

[pcAlgo](#) 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)
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]
pcSelect(d.mat[,10],d.mat[,-10],alpha=0.05)
## You see, that variable 4,5,6 are considered as important
## By inspecting zMin 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
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.presele`      *PC-Select preselection: 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.presele(y, dm, alpha, alphapre, corMethod = "standard", verbose = 0, directed
```

### Arguments

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

### Details

This function basically applies `pcAlgo` 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 quite a lot.

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

### Value

<code>pcs</code>	A boolean vector indicating which column of <code>dm</code> is associated with <code>y</code>
<code>zMin</code>	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.
<code>Xnew</code>	Preselected Variables.

### Author(s)

Philipp Ruetimann.

## References

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

## See Also

[pcAlgo](#) 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)
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.preset(d.mat[,10], d.mat[,-10], alpha=0.05, alphapre=0.6)
```

---

pdag2dag

*Extend a 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 PDGA (graph object)
<code>keepVstruct</code>	If TRUE, the v-structures in <code>g</code> are kept. O/w they are ignored and an arbitrary extension is generated.

## Details

The PDAG is consistently extended using an algorithm by Dor and Tarsi (see References). 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 consistent DAG extension (graph object) and "success" is TRUE iff the extension was possible.

**Author(s)**

Markus Kalisch ([kalisch@stat.math.ethz.ch](mailto:kalisch@stat.math.ethz.ch))

**References**

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

**See Also**

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

**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)
```

---

plotAG

*Plot partial ancestral graphs (PAG)*

---

**Description**

This function plots partial ancestral graphs, namely graphs with the following six kinds of edges: -, ->, <->, o-, o-o, o->.

**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;  $M[i,j]=1, M[j,i] \neq 0$ :  $i \rightarrow j$ ;  $M[i,j]=2, M[j,i] \neq 0$ :  $i \leftarrow j$ ;  $M[i,j]=3, M[j,i] \neq 0$ :  $i \leftrightarrow j$ .

**Details**

Used to plot the output of the function `udag2pag`. Note that Rgraphviz places the edge marks sometimes not quite at the end of the edge. This is just a layout problem and should not hinder a correct interpretation.

**Author(s)**

Markus Kalisch ([kalisch@stat.math.ethz.ch](mailto:kalisch@stat.math.ethz.ch))

**See Also**

[pcAlgo](#)

**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, 0,0,0,1,0),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")
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)
resD

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

```

resP <- udag2pag(resD, rules=rules, verbose=1)
colnames(resP) <- rownames(resP) <- nodes(g)[-L]

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

```

---

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](mailto: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**

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

---

 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.

**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 <a href="#">tsort</a> from the <b>RBGL</b> 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](mailto:kalisch@stat.math.ethz.ch)) and Martin Maechler.

## See Also

[randomDAG](#) for generating a random DAG; [pcAlgo](#) for estimating the skeleton 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)

## 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, Vol. 65, 31-78.

**See Also**

[pcAlgo](#)

**Examples**

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

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)
```

**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 <code>i</code> means that rule <code>i</code> ( $R_i$ ) will be used. Per default, all rules are set to TRUE.
<code>verbose</code>	0: No output; 1: Details. Default is 0.

**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](mailto: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, pp. 1873-1896

**See Also**

[pcAlgo](#)

**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")
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)
resD

## extend the pcalgo-object into a PAG using all 10 rules
rules <- rep(TRUE,10)
resP <- udag2pag(resD, rules=rules, verbose=1)
colnames(resP) <- rownames(resP) <- nodes(g)[-L]

## 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 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**

```
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](mailto: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")

## estimate skeleton
resU <- pcAlgo(d.mat, alpha = 0.05, corMethod = "standard", directed=FALSE)
## 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)
```

### 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). There is no test whether the result is really extendible.

### Value

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

### Author(s)

Markus Kalisch ([kalisch@stat.math.ethz.ch](mailto: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")

## estimate skeleton
resU <- pcAlgo(d.mat, alpha = 0.05, corMethod = "standard", directed=FALSE)
## 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 extendable
xtbl	Is final graph with v-structure extendable
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 extendable; 1: reorienting double edge visits helps; 2: orig. try is not extendable; 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](mailto: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")

## estimate skeleton
resU <- pcAlgo(d.mat, alpha = 0.05, corMethod = "standard", directed=FALSE)
## 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, 12
- \*Topic **classes**
  - pcAlgo-class, 17
- \*Topic **datagen**
  - randomDAG, 25
  - rmvDAG, 26
- \*Topic **graphs**
  - beta.special, 1
  - beta.special.pcObj, 3
  - compareGraphs, 5
  - corGraph, 8
  - dag2cpdag, 9
  - decHeur, 10
  - pcAlgo, 14
  - pcSelect, 19
  - pcSelect.preset, 21
  - pdag2dag, 22
  - plotAG, 23
  - randomDAG, 25
  - shd, 28
  - udag2pag, 29
  - udag2pdag, 30
  - udag2pdagRelaxed, 32
  - udag2pdagSpecial, 33
- \*Topic **htest**
  - condIndFisherZ, 6
- \*Topic **models**
  - beta.special, 1
  - beta.special.pcObj, 3
  - corGraph, 8
  - dag2cpdag, 9
  - decHeur, 10
  - pcAlgo, 14
  - pcSelect, 19
  - pcSelect.preset, 21
  - plotAG, 23
  - udag2pag, 29
  - udag2pdag, 30
  - udag2pdagRelaxed, 32
  - udag2pdagSpecial, 33
- \*Topic **multivariate**
  - beta.special, 1
  - beta.special.pcObj, 3
  - condIndFisherZ, 6
  - dag2cpdag, 9
  - decHeur, 10
  - mcor, 13
  - pcAlgo, 14
  - pcorOrder, 18
  - pcSelect, 19
  - pcSelect.preset, 21
  - plotAG, 23
  - rmvDAG, 26
  - udag2pag, 29
  - udag2pdag, 30
  - udag2pdagRelaxed, 32
  - udag2pdagSpecial, 33
- \*Topic **robust**
  - decHeur, 10
  - mcor, 13
- \*Topic **utilities**
  - getNextSet, 12
- beta.special, 1
- beta.special.pcObj, 2, 3, 4
- class, 16
- combn, 12
- compareGraphs, 5, 16, 26
- condIndFisherZ, 6, 14, 19
- corGraph, 8
- covOGK, 13, 14
- dag2cpdag, 2, 4, 9, 31, 32, 34
- decHeur, 10
- getNextSet, 12
- graph-class, 16, 25

logical, 7

mcor, 13, 15

pcAlgo, 2, 4, 11, 14, 16–18, 20–22, 24, 27–29

pcAlgo-class, 17

pcorOrder, 8, 14, 18

pcSelect, 19

pcSelect.presel, 21

pdag2dag, 10, 22, 23, 31, 32, 34

plot, pcAlgo-method  
(pcAlgo-class), 17

plotAG, 23

Qn, 13, 14

qnorm, 7

randomDAG, 6, 16, 25, 27

randomGraph, 16

rmvDAG, 16, 26, 26

shd, 28

show, pcAlgo-method  
(pcAlgo-class), 17

summary, pcAlgo-method  
(pcAlgo-class), 17

tsort, 26

udag2pag, 29

udag2pdag, 10, 16, 23, 30, 31, 32, 34

udag2pdagRelaxed, 31, 32, 32, 34

udag2pdagSpecial, 31, 32, 33

ugraph, 14

zStat (condIndFisherZ), 6