The ggm Package

November 15, 2003

Version 0.8-1

Date 2003-11-14

Title Graphical Gaussian Models

Author Giovanni M. Marchetti, Mathias Drton

Maintainer Giovanni M. Marchetti <gmm@ds.unifi.it>

Depends R (>= 1.8.0)

Description Functions for fitting Gaussian Markov models.

License GPL version 2 or newer

R topics documented:

`DAG` *Directed acyclic graphs (DAGs)*

Description

A simple way to define a DAG by means of regression model formulae.

Usage

```
DAG(..., order = FALSE)
```

Arguments

`...`	a sequence of model formulae
`order`	logical, defaulting to `FALSE`. If `TRUE` the nodes of the DAG are permuted according to the topological order. If `FALSE` the nodes are in the order they first appear in the model formulae (from left to right).

Details

The DAG is defined by a sequence of recursive regression models. Each regression is defined by a model formula. For each formula the response defines a node of the graph and the explanatory variables the parents of that node. If the regressions are not recursive the function returns an error message.

Some authors prefer the terminology acyclic directed graphs (ADG).

Value

the adjacency matrix of the DAG, i.e. a square Boolean matrix of order equal to the number of nodes of the graph and a one in position (i, j) if there is an arrow from i to j and zero otherwise. The rownames of the adjacency matrix are the nodes of the DAG.

If `order = TRUE` the adjacency matrix is permuted to have parents before children. This can always be done (in more than one way) for DAGs. The resulting adjacency matrix is upper triangular.

Note

The model formulae may contain interactions, but they are ignored in the graph.

Author(s)

G. M. Marchetti

References

Lauritzen, S. (1996). *Graphical models.* Oxford: Clarendon Press.

See Also

UG, topSort, edgeMatrix, fitDag

Examples

```
## A Markov chain
DAG(y ~ x, x ~ z, z ~ u)

## Another DAG
DAG(y ~ x + z + u, x ~ u, z ~ u)

## A DAG with an isolated node
DAG(v ~ v, y ~ x + z, z ~ w + u)

## There can be repetitions
DAG(y ~ x + u + v, y ~ z, u ~ v + z)

## Interactions are ignored
DAG(y ~ x*z + z*v, x ~ z)

## A cyclic graph returns an error!
## Don't run: DAG(y ~ x, x ~ z, z ~ y)

## The order can be changed
DAG(y ~ z, y ~ x + u + v,  u ~ v + z)

## If you want to order the nodes (topological sort of the DAG)
DAG(y ~ z, y ~ x + u + v,  u ~ v + z, order=TRUE)
```

In *Indicator matrix*

Description

Finds the indicator matrix of the zeros of a matrix.

Usage

```
In(A)
```

Arguments

`A`	a matrix.

Details

The indicator matrix is a matrix of zeros and ones which has a zero element iff the corresponding element of `A` is (exactly) zero.

Value

a matrix of the same dimensions as `A`.

Author(s)

Giovanni M. Marchetti

References

Wermuth, N. & Cox, D.R. (2003). Joint response graphs and separation induced by triangular systems. Submitted and available at `http://psystat.sowi.uni-mainz.de`.

See Also

`DAG`, `inducedCovGraph`, `inducedConGraph`

Examples

```
## A simple way to find the overall induced coincentration graph
## The DAG on p. 198 of Cox & Wermuth (1996)
amat <- DAG(y1 ~ y2 + y3, y3 ~ y5, y4 ~ y5)
A <- edgeMatrix(amat)
In(crossprod(A))
```

`InducedGraphs` *Graphs induced by marginalization or conditioning*

Description

Functions to find the induced covariance or concentration graphs after conditioning on a set of variables and marginalizing over another set.

Usage

```
inducedCovGraph(amat, sel = rownames(amat), cond = NULL)
inducedConGraph(amat, sel = rownames(amat), cond = NULL)
```

Arguments

`amat`	a square Boolean matrix, the adjacency matrix of a directed acyclic graph. The names of rows and of the columns are the nodes of the DAG.
`sel`	a character vector representing a subset of selected variables. The elements of the vector must be a subset of the names of the nodes i.e. of `rownames(A)`. By default `sel` is the set of the nodes of the DAG.
`cond`	a character vector representing the variables on which you want to condition. `cond` must be disjoint from `sel` and their union must be a subset of the set of nodes. The set difference between the set of nodes and the union of `sel` and `cond` are the variables over which we marginalize. `cond` may be the null vector (the default), meaning that you want to condition on the empty set.

Details

Given a directed acyclic graph representing a set of conditional independencies it is possible to obtain other graphs of conditional independence implied after marginalizing over and conditionig on sets of nodes. Two such graphs are the covariance graph and the concentration graph (Cox & Wermuth, 1996, 2003).

Value

`inducedCovGraph` returns the adjacency matrix of the covariance graph of the variables in set `sel` given the variables in set `cond`, implied by the original directed acyclic graph with adjacency matrix `amat`.

`inducedConGraph` returns the adjacency matrix of the concentration graph of the variables in set `sel` given the variables in set `cond`, implied by the original directed acyclic graph with adjacency matrix `amat`.

If `sel` is `NULL` the functions return the null matrix. If `cond` is `NULL`, the conditioning set is empty and the functions return the overall induced covariance or concentration matrices of the selected variables.

Note

If you do not specify `sel` you cannot specify a non `NULL` value of `cond`.

Author(s)

Giovanni M. Marchetti

References

Cox, D. R. & Wermuth, N. (1996). *Multivariate dependencies*. London: Chapman & Hall.

Wermuth, N. & Cox, D.R. (2003). Joint response graphs and separation induced by triangular systems. Submitted and available at http://psystat.sowi.uni-mainz.de.

See Also

DAG, UG,isAcyclic

Examples

```
## Define a DAG
dag <- DAG(a ~ x, c ~ b+d, d~ x)
dag
## Induced covariance graph of a, b, d given the empty set.
inducedCovGraph(dag, sel=c("a", "b", "d"), cond=NULL)

## Induced concentration graph of a, b, c given x
inducedConGraph(dag, sel=c("a", "b", "c"), cond="x")

## Overall covariance graph
inducedCovGraph(dag)

## Overall concentration graph
inducedConGraph(dag)

## Induced covariance graph of x, b, d given c, x.
inducedCovGraph(dag, sel=c("a", "b", "d"), cond=c("c", "x"))

## Induced concentration graph of a, x, c given d, b.
inducedConGraph(dag, sel=c("a", "x", "c"), cond=c("d", "b"))

## The DAG on p. 198 of Cox & Wermuth (1996)
dag <- DAG(y1~ y2 + y3, y3 ~ y5, y4 ~ y5)

## Cf. figure 8.7 p. 203 in Cox & Wermuth (1996)
inducedCovGraph(dag, sel=c("y2", "y3", "y4", "y5"), cond="y1")
inducedCovGraph(dag, sel=c("y1", "y2", "y4", "y5"), cond="y3")
inducedCovGraph(dag, sel=c("y1", "y2", "y3", "y4"), cond="y5")

## Cf. figure 8.8 p. 203 in Cox & Wermuth (1996)
inducedConGraph(dag, sel=c("y2", "y3", "y4", "y5"), cond="y1")
inducedConGraph(dag, sel=c("y1", "y2", "y4", "y5"), cond="y3")
inducedConGraph(dag, sel=c("y1", "y2", "y3", "y4"), cond="y5")

## Cf. figure 8.9 p. 204 in Cox & Wermuth (1996)
inducedCovGraph(dag, sel=c("y2", "y3", "y4", "y5"), cond=NULL)
inducedCovGraph(dag, sel=c("y1", "y2", "y4", "y5"), cond=NULL)
inducedCovGraph(dag, sel=c("y1", "y2", "y3", "y4"), cond=NULL)

## Cf. figure 8.10 p. 204 in Cox & Wermuth (1996)
inducedConGraph(dag, sel=c("y2", "y3", "y4", "y5"), cond=NULL)
inducedConGraph(dag, sel=c("y1", "y2", "y4", "y5"), cond=NULL)
inducedConGraph(dag, sel=c("y1", "y2", "y3", "y4"), cond=NULL)
```

`Simple Graph Operations` *Simple graph operations*

Description

Finds the boundary, children, parents of a subset of nodes of a graph.

Usage

```
bd(nn, amat)
ch(nn, amat)
pa(nn, amat)
```

Arguments

`nn`	a vector of nodes. It may either a numeric vector, or a character vector. If it is character vector must be a subset of the `rownames` of the edge matrix.
`amat`	a square matrix with dimnames specifying the adjacency matrix of the graph

Details

For definitions of these operators see Lauritzen (1996).

Value

A vector, specifying the boundary or the children or the parents of nodes `nn` in the graph. This is a numeric or a character vector depending on the mode of `nn`.

Author(s)

Giovanni M. Marchetti

References

Lauritzen, S. (1996). *Graphical models.* Oxford: Clarendon Press.

See Also

`UG`, `DAG`

Examples

```
## find boundary of a subset of nodes of a DAG
G <- DAG(y ~ x+b+a, b~a, x~a)
bd("b", G)
bd(c("b", "x"), G)
bd("x", G)
bd(c("x","b"), G)
## find boundary of a subset of nodes of an UG
G <- UG(~ y*x*z + z*h*v)
bd("z", G)
bd(c("y", "x"), G)
bd("v", G)
bd(c("x","v"), G)
## children of a subset of nodes of a DAG
G <- DAG(y ~ x+b+a, b~a, x~a)
ch("b", G)
ch(c("b", "x"), G)
ch("x", G)
ch(c("a","x"), G)
## parents of a subset of nodes of a DAG
pa("b", G)
pa(c("b", "x"), G)
pa("x", G)
pa(c("x","b"), G)
```

`UG` *Defining an undirected graph (UG)*

Description

A simple way to define an undirected graph by means of a single model formula.

Usage

```
UG(f)
```

Arguments

`f`	a single model formula without response

Details

The undirected graph $G = (V, E)$ is defined by a set of nodes V and a set of pairs E. The set of pairs is defined by the set of interactions in the formula. Interactions define complete subgraphs (not necessarily maximal) of the UG. The best way is to specify interactions that match the cliques of the undirected graph. This is the standard way to define graphical models for contingency tables. Remember that some hierarchical models are not graphical, but they imply the same graph.

The function returns the edge matrix of the graph, i.e. a square Boolean matrix of order equal to the number of nodes of the graph and a one in position (i, j) if there is an arrow from j to i and zero otherwise. By default this matrix has ones along the main diagonal. For UGs this matrix is symmetric. The dimnames of the edge matrix are the nodes of the UG.

Value

a Boolean matrix with dimnames, the adjacency matrix of the undirected graph.

Note

Author(s)

Giovanni M. Marchetti

References

Lauritzen, S. (1996). *Graphical models*. Oxford: Clarendon Press.

See Also

`fitConGraph`, `fitCovGraph`

Examples

```
## X independent of Y given Z
UG(~ X*Z + Y*Z)

# The saturated model
UG(~ X*Y*Z)

## The model without three-way interactions has the same graph
UG(~ X*Y + Y*Z + Z*X)
UG(~ (X + Y + Z)^2)

## Butterfly model defined from the cliques
UG(~ mec*vec*alg + alg*ana*sta)

## Some isolated nodes
UG(~x*y*z + a + b)
```

`adjMatrix` *Adjacency matrix of a graph*

Description

Transforms the "edge matrix" of a graph into the adjacency matrix.

Usage

```
adjMatrix(A)
```

Arguments

`A`	a square matrix representing the edge matrix of a graph.

Details

Given the edge matrix A of a graph, this can be transformed into an adjacency matrix E with the formula $E = (A - I)^T$.

Value

`E`	the adjacency matrix of the graph.

Author(s)

Giovanni M. Marchetti

See Also

`edgeMatrix`

Examples

```
amat <- DAG(y ~ x+z, z~u+v)
E <- edgeMatrix(amat)
adjMatrix(E)
```

`allEdges` *All edges of a graph*

Description

Finds the set of edges of a graph. That is the set of undirected edges if the graph is undirected and the set of arrows if the graph is directed.

Usage

```
allEdges(amat)
```

Arguments

`amat`	a square Boolean matrix, with dimnames, the adjacency matrix of a graph.

Value

a matrix with two columns. Each row of the matrix is a pair of indices indicating an edge of the graph. If the graph is undirected, then only one of the pairs $(i, j), (j, i)$ is reported.

Author(s)

Giovanni M. Marchetti

See Also

`cycleMatrix`

Examples

```
## A UG graph
allEdges(UG(~ y*v*k +v*k*d+y*d))

## A DAG
allEdges(DAG(u~h+o+p, h~o, o~p))
```

---

basiSet *Basis set of a DAG*

Description

Finds a basis set for the conditional independencies implied by a directed acyclic graph, that is a minimal set of independencies that imply all the other ones.

Usage

```
basiSet(amat)
```

Arguments

`amat`	a square matrix with dimnames representing the adjacency matrix of a DAG.

Details

Given a DAG and a pair of non adjacent nodes (i, j) such that j has higher causal order than i, the set of independency statements i independent of j given the union of the parents of both i and j is a basis set (see Shipley, 2000). This basis set has the property to lead to independent test statistics.

Value

a list of vectors representing several conditional independence statements. Each vector contains the names of two non adjacent nodes followed by the names of nodes in the conditioning set (which may be empty).

Author(s)

Giovanni M. Marchetti

References

Shipley, B. (2000). A new inferential test for path models based on directed acyclic graphs. *Structural Equation Modeling*, 7(2), 206–218.

See Also

`shipley.test`, `dSep`, `DAG`

Examples

```
## See Shipley (2000), Figure 2, p. 213
A <- DAG(x5~ x3+x4, x3~ x2, x4~x2, x2~ x1)
basiSet(A)
```

`bfs` *Breadth first search*

Description

Breadth-first search of a connected undirected graph.

Usage

```
bfs(amat, v = 1)
```

Arguments

`amat`	a symmetric matrix with dimnames specifying the adjacency matrix of the undirected graph
`v`	an integer, indicating the starting node of the search. Defaults to the first node.

Details

Breadth-first search is a systematic method for exploring a graph. The algorithm is taken from Aho, Hopcroft & Ullman (1983).

Value

`tree`	the edge matrix of the resulting spanning tree
`branches`	a matrix with two columns, giving the indices of the branches of the spanning tree
`chords`	a matrix with two columns, giving the indices of the chords of the spanning tree

Author(s)

Giovanni M. Marchetti

References

Aho, A.V., Hopcrtoft, J.E. & Ullman, J.D. (1983). *Data structures and algorithms.* Reading: Addison-Wesley.

Thulasiraman, K. & Swamy, M.N.S. (1992). *Graphs: theory and algorithms.* New York: Wiley.

See Also

`UG`, `findPath`, `cycleMatrix`

Examples

```
## Finding a spanning tree of the butterfly graph
bfs(UG(~ a*b*o + o*u*j))
## Starting from another node
bfs(UG(~ a*b*o + o*u*j), v=3)
```

`checkIdent` *Identifiability of a model with one latent variable*

Description

Checks four sufficient conditions for identifiability of a Gaussian DAG model with one latent variable.

Usage

```
checkIdent(amat, latent)
```

Arguments

`amat`	a square matrix with dimnames, representing the adjacency matrix of a DAG.
`latent`	an integer representing the latent variables among the nodes, or the name of the node.

Details

Stanghellini and Wermuth (2003) give some sufficient conditions for checking if a Gaussian model that factorizes according to a DAG is identified when there is one hidden node over which we marginalize. Specifically, the function checks the conditions of Theorem 1, (i) and (ii) and of Theorem 2 (i) and (ii).

Value

a vector of length four, indicating if the model is identified according to the conditions of theorems 1 and 2 in Stanghellini & Wermuth (2003). The answer is `TRUE` if the condition holds and thus the model is globally identified or `FALSE` if the condition fails, and thus we do not know if the model is identifiable.

Author(s)

Giovanni M. Marchetti

References

Stanghellini, E. & Wermuth, N. (2003). On the identification of path-analysis models with one hidden variable. Submitted and available at `http://psystat.sowi.uni-mainz.de`.

See Also

`isGident`, `InducedGraphs`

Examples

```
## See DAG in Figure 4 (a) in Stanghellini & Wermuth (2003)
d <- DAG(y1 ~ y3, y2 ~ y3 + y5, y3 ~ y4 + y5, y4 ~ y6)
checkIdent(d, "y3")  # Identifiable
checkIdent(d, "y4")  # Not identifiable?

## See DAG in Figure 5 (a) in Stanghellini & Wermuth (2003)
d <- DAG(y1 ~ y5+y4, y2 ~ y5+y4, y3 ~ y5+y4)
checkIdent(d, "y4")  # Identifiable
checkIdent(d, "y5")  # Identifiable

## A simple function to check identifiability for each node

is.ident <- function(amat){
### Check suff. conditions on each node of a DAG.
   p <- nrow(amat)
   ## Degrees of freedom
     df <- p*(p+1)/2 - p  - sum(amat==1) - p + 1
   if(df <= 0)
       warning(paste("The degrees of freedom are ", df))
    a <- rownames(amat)
    for(i in a) {
      b <- checkIdent(amat, latent=i)
      if(TRUE %in% b)
        cat("Node", i, names(b)[!is.na(b)], "\n")
      else
        cat("Unknown.\n")
    }
  }
```

---

cliques   *Cliques of an undirected graph*

Description

Finds the cliques of an undirected graph.

Usage

```
cliques(amat)
```

Arguments

`amat`	a square Boolean matrix with dimnames, representing the adjacency matrix of an undirected graph.

Details

The cliques of a graph are the subsets of nodes which induce a maximally complete subgraph. Determining the cliques of a graph is NP-hard in general.

Value

a list of vectors of nodes.

Author(s)

Mathias Drton

References

Lauritzen, S. (1996). *Graphical models.* Oxford: Clarendon Press.

See Also

UG, bd

Examples

```
u <- UG(~ a*b*c + c*d*e*g)
u
cliques(u)

graph22 <- UG(~x1*x2*x3+x3*x4*x5*x6+x5*x7*x8*x9*x10+x9*x11*x12*x13*x14+
x12*x15+x15*x16*x17*x18+x17*x1*x18+x1*x5*x7*x19*x20+x6*x20*x21+x22)

cliques(graph22)
```

---

cmpGraph *The complementary graph*

Description

Finds the complementary graph of an undirected graph.

Usage

```
cmpGraph(amat)
```

Arguments

`amat`	the adjacency matrix of an undirected graph

Details

The complementary graph of an UG is the graph that has the same set of nodes and an undirected edge connecting i and j whenever there is not an (i, j) edge in the original UG.

Value

the edge matrix of the complementary graph.

Author(s)

Giovanni M. Marchetti

References

Lauritzen, S. (1996). *Graphical models.* Oxford: Clarendon Press.

See Also

UG, DAG

Examples

```
## A chordless four-cycle
four <- UG(~ a*b + b*d + d*e + e*a)
four
cmpGraph(four)
```

---

conComp *Connectivity components*

Description

Finds the connectivity components of a graph.

Usage

```
conComp(amat)
```

Arguments

`amat`	a square matrix with dimnames, the adjacency matrix of a DAG or a UG.

Value

an integer vector representing a partition of the set of nodes.

Author(s)

Giovanni M. Marchetti

References

Lauritzen, S. (1996). *Graphical models.* Oxford: Clarendon Press.

See Also

UG

Examples

```
## three connected components
conComp(UG(~a*c+c*d+e+g*o*u))
## a connected graph
conComp(UG(~ a*b+b*c+c*d+d*a))
```

`correlations` *Marginal and partial correlations*

Description

Computes a correlation matrix with ones along the diagonal, marginal correlations in the lower triangle and partial correlations given all remaining variables in the upper triangle.

Usage

```
correlations(x)
```

Arguments

`x`	a square symmetric matrix, a covariance matrix, or a data.frame for n observations and p variables.

Value

a square correlation matrix with marginal correlations (lower triangle) and partial correlations (upper triangle).

Author(s)

Giovanni M. Marchetti

References

Cox, D. R. & Wermuth, N. (1996). *Multivariate dependencies.* London: Chapman & Hall.

See Also

`parcor`, `cor`

Examples

```
## See Table 6.1 in Cox & Wermuth (1996)
data(glucose)
correlations(glucose)
```

`cycleMatrix` *Fundamental cycles*

Description

Finds the matrix of fundamental cycles of a connected undirected graph.

Usage

```
cycleMatrix(amat)
```

Arguments

`amat`	a symmetric matrix with dimnames denoting the adjacency matrix of the undirected graph. The graph must be connected, otherwise the function returns an error message.

Details

All the cycles in an UG can be obtained from combination (ring sum) of the set of fundamental cycles. The matrix of fundamental cycles is a Boolean matrix having as rows the fundamental cycles and as columns the edges of the graph. If an entry is one then the edge associated to that column belongs to the cycle associated to the row.

Value

a Boolean matrix of the fundamental cycles of the undirected graph. If there is no cycle the function returns `NULL`.

Note

This function is used by `isGident`. The row sum of the matrix gives the length of the cycles.

Author(s)

Giovanni M. Marchetti

References

Thulasiraman, K. & Swamy, M.N.S. (1992). *Graphs: theory and algorithms.* New York: Wiley.

See Also

`UG`, `findPath`, `fundCycles`, `isGident`, `bfs`

Examples

```
## Three cycles
cycleMatrix(UG(~a*b*d+d*e+e*a*f))
## No cycle
 cycleMatrix(UG(~a*b))
## two cycles: the first is even and the second is odd
cm <- cycleMatrix(UG(~a*b+b*c+c*d+d*a+a*u*v))
apply(cm, 1, sum)
```

`dSep` *d-separation*

Description

Determines if in a directed acyclic graph two set of nodes a d-separated by a third set of nodes.

Usage

```
dSep(amat, first, second, cond)
```

Arguments

`amat`	a Boolean matrix with dimnames, representing the adjacency matrix of a directed acyclic graph. The function does not check if this is the case. See the function `isAcyclic`.
`first`	a vector representing a subset of nodes of the DAG. The vector should be a character vector of the names of the variables matching the names of the nodes in `rownames(A)`. It can be also a numeric vector of indices.
`second`	a vector representing another subset of nodes of the DAG. The set `second` must be disjoint from `first`. The mode of `second` must match the mode of `first`.
`cond`	a vector representing a conditioning subset of nodes. The set `cond` must be disjoint from the other two sets and must share the same mode.

Details

d-separation is a fundamental concept introduced by Pearl (1988).

Value

a logical value. `TRUE` if `first` and `second` are d-separated by `cond`.

Author(s)

Giovanni M. Marchetti

References

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems.* San Mateo: Morgan Kaufmann.

Lauritzen, S. (1996). *Graphical models.* Oxford: Clarendon Press.

See Also

`DAG`, `shipley.test`, `inducedCovGraph`

Examples

```
## Conditioning on a transition node
dSep(DAG(y ~ x, x ~ z), first="y", second="z", cond = "x")
## Conditioning on a collision node (collider)
dSep(DAG(y ~ x, y ~ z), first="x", second="z", cond = "y")
## Conditioning on a source node
dSep(DAG(y ~ x, z ~ x), first="y", second="z", cond = "x")
## Marginal independence
dSep(DAG(y ~ x, y ~ z), first="x", second="z", cond = NULL)
## The DAG defined on p.~47 of Lauritzen (1996)
dag <- DAG(g ~ x, h ~ x+f, f ~ b, x ~ l+d, d ~ c, c ~ a, l ~ y, y ~ b)
dSep(dag, first="a", second="b", cond=c("x", "y"))
dSep(dag, first="a", second=c("b", "d"), cond=c("x", "y"))
```

`drawGraph` *Drawing a graph with a simple point and click interface.*

Description

Draw a graph from its adjacency matrix representation.

Usage

```
drawGraph(amat, coor = NULL, bid = TRUE, adjust = TRUE, alpha = 1, beta = 3, lwd = 1)
```

Arguments

`amat`	the adjacency matrix representation of the graph. This can be an undirected graph a directed acyclic graph or an ancestral graph.
`coor`	an optional matrix of dimensions $p \times 2$ where p is the number of vertices of the graph.If `coor=NULL` then the function chooses a default position for the nodes.
`bid`	a logical value indicating if the edges of a covariance graph must be plotted as bidirected (`bid=TRUE`)or as dashed lines (`bid=FALSE`).
`adjust`	a logical value. If `TRUE` the graph is plotted and the system waits until the mouse button is pressed (same behaviour of `locator` function.
`alpha`	a positive value between controlling the distance from the end of the edges to the nodes of the graph.
`beta`	a positive value controlling the distance of the labels of the variables from the nodes.
`lwd`	line width of the edges.

Details

The function is a very simple tool useful for displaying small graphs, with a rudimentary interface for moving nodes and edges of a given graph and adjusting the final plot. For better displays use **Rgraphviz** package in Bioconductor project.

Value

The function plots the graph with a initial positioning of the nodes, as specified by `coor` and remains in a waiting state. The position of each node can be shifted by pointing and clicking (with the first mouse button) close to the node. When the mouse button is pressed the node which is closer to the selected point is moved to that position. Thus, one must be careful to click closer to the selected node than to any other node. The nodes can be moved to any position by repeating the previous operation. The adjustment process is terminated by pressing any mouse button other than the first.

At the end of the process, the function returns invisibly the coordinates of the nodes. The coordinates may be used later to redisplay the graph.

Author(s)

Giovanni M. Marchetti

References

Rgraphwiz, `http://www.bioconductor.org`.

GraphViz, Graph Visualization Project. AT&T Research. `http://www.graphviz.org`.

See Also

`UG`, `DAG`, `makeAG`

Examples

```
## A directed acyclic graph
d <- DAG(y1 ~ y2+y6, y2 ~ y3, y3 ~ y5+y6, y4 ~ y5+y6)
## Don't run: drawGraph(d)

## An undirected graph
g <- UG(~giova*anto*armo + anto*arj*sara)
## Don't run: drawGraph(d)

## An ancestral graph
ag <- makeAG(ug=UG(~y0*y1), dag=DAG(y4~y2, y2~y1), bg=UG(~y2*y3+y3*y4))
## Don't run: drawGraph(ag)
## Don't run: drawGraph(ag, bid=FALSE)

## A more complex example with coordinates: the UNIX evolution
xy <-
structure(c(5, 15, 23, 25, 26, 17, 8, 6, 6, 7, 39, 33, 23, 49,
19, 34, 13, 29, 50, 68, 70, 86, 89, 64, 81, 45, 64, 49, 64, 87,
65, 65, 44, 37, 64, 68, 73, 85, 83, 95, 84, 0, 7, 15, 27, 44,
37, 36, 20, 51, 65, 44, 64, 59, 73, 69, 78, 81, 90, 97, 89, 72,
85, 74, 62, 68, 59, 52, 48, 43, 50, 34, 21, 18, 5, 1, 10, 2,
11, 2, 1, 44), .Dim = c(41, 2), .Dimnames = list(NULL, c("x",
"y")))
Unix <- DAG(
                SystemV.3 ~ SystemV.2,
                SystemV.2 ~ SystemV.0,
                SystemV.0 ~ TS4.0,
                TS4.0 ~ Unix.TS3.0 + Unix.TS.PP + CB.Unix.3,
                PDP11.SysV ~ CB.Unix.3,
                CB.Unix.3 ~ CB.Unix.2,
                CB.Unix.2 ~ CB.Unix.1,
                Unix.TS.PP ~ CB.Unix.3,
                Unix.TS3.0 ~ Unix.TS1.0 + PWB2.0 + USG3.0 + Interdata,
                USG3.0 ~ USG2.0,
                PWB2.0 ~ Interdata + PWB1.2,
                USG2.0 ~ USG1.0,
                CB.Unix.1 ~ USG1.0,
                PWB1.2 ~ PWB1.0,
                USG1.0 ~ PWB1.0,
                PWB1.0 ~ FifthEd,
                SixthEd ~ FifthEd,
                LSX ~ SixthEd,
                MiniUnix ~ SixthEd,
                Interdata ~ SixthEd,
                Wollongong ~ SixthEd,
                SeventhEd ~ Interdata,
                BSD1 ~ SixthEd,
                Xenix ~ SeventhEd,
                V32 ~ SeventhEd,
                Uniplus ~ SeventhEd,
                BSD3 ~ V32,
                BSD2 ~ BSD1,
                BSD4 ~ BSD3,
                BSD4.1 ~ BSD4,
                EigthEd ~ SeventhEd + BSD4.1,
                NinethEd ~ EigthEd,
                Ultrix32 ~ BSD4.2,
                BSD4.2 ~ BSD4.1,
                BSD4.3 ~ BSD4.2,
                BSD2.8 ~ BSD4.1 + BSD2,
                BSD2.9 ~ BSD2.8,
                Ultrix11 ~ BSD2.8 + V7M + SeventhEd,
                V7M ~ SeventhEd
                )
drawGraph(Unix, coor=xy, adjust=FALSE)
# dev.print(file="unix.fig", device=xfig) # Edit the graph with Xfig
```

---

edgeMatrix *Edge matrix of a graph*

Description

Transforms the adjacency matrix of a graph into an “edge matrix”.

Usage

```
edgeMatrix(E, inv=FALSE)
```

Arguments

`E`	a square matrix, representing the adjacency matrix of a graph.
`inv`	a logical value.

Details

In some matrix computations for graph objects the adjacency matrix of the graph is transformed into an "edge matrix". Briefly, if E is the adjacency matrix of the graph, the edge matrix is $A = (E + I)^T = [a_{ij}]$. Thus, A has ones along the diagonal and if the graph has no edge beteween nodes i and j the entries $a_{i,j}$ and $a_{j,i}$ are both zero. If there is an arrow from j to i $a_{i,j} = 1$ and $a_{j,i} = 0$. If there is an undirected edge, both $a_{i,j} = a_{j,i} = 1$.

Value

`A`	the edge matrix of the graph. If `TRUE` the nodes are sorted in inverted topological order and the edge matrix is upper triangular.

Author(s)

Giovanni M. Marchetti

References

Wermuth, N. (2003). Analysing social science data with graphical Markov models. In: *Highly Structured Stochastic Systems.* P. Green, N. Hjort & T. Richardson (eds.), 47–52. Oxford: Oxford University Press.

See Also

`adjMatrix`

Examples

```
amat <- DAG(y ~ x+z, z~u+v)
amat
edgeMatrix(amat)
edgeMatrix(amat, inv=TRUE)
```

`findPath` *Finding paths*

Description

Finds one path between two nodes of a graph.

Usage

```
findPath(amat, st, en, path = c())
```

Arguments

`amat`	a square Boolean matrix with dimnames, the adjacency matrix of a graph.
`st`	an integer, the starting node.
`en`	an integer, the ending node.
`path`	a vector of integers, used in recursive calls. At the beginning is `NULL`. It should not be modified by the user.

Value

a vector of integers, the sequence of nodes of a path, starting from `st` to `en`. In some graphs (spanning trees) there is only one path between two nodes.

Note

This function is not intended to be directly called by the user.

Author(s)

Giovanni M. Marchetti, translating the original **Python** code (see references).

References

Python Softftware Foundation (2003). Python Patterns — Implementing Graphs. `http://www.python.org/docs/essays/graphs.html`.

See Also

`fundCycles`

Examples

```
## A (single) path on a spanning tree
findPath(bfs(UG(~ a*b*c + b*d + d*e+ e*c))$tree, st=1, en=5)
```

`fitAncestralGraph` *Fitting of Gaussian Ancestral Graph Models*

Description

Iterative conditional fitting of Gaussian Ancestral Graph Models.

Usage

```
fitAncestralGraph(amat, S, n, tol = 1e-06)
```

Arguments

`amat`	a square matrix, representing the adjacency matrix of an ancestral graph.
`S`	a symmetric positive definite matrix with dimnames, the sample covariance matrix.
`n`	the sample size, a positive integer.
`tol`	a small positive number indicating the tolerance used in convergence checks.

Details

Ancestral graph models were introduced by Richardson & Spirtes (2002) as a class of graphical models whose global Markov property is closed under conditioning and marginalization. In the Gaussian case, the models can be parameterized using precision parameters, regression coefficients, and error covariances (compare Richardson & Spirtes, 2002, Section 8). This function finds the MLE $\hat{\Lambda}$ of the precision parameters by fitting a concentration graph model. The MLE $\hat{B}$ of the regression coefficients and the MLE $\hat{\Omega}$ of the error covariances are obtained by iterative conditional fitting (Drton & Richardson, 2003a, b). The three sets of parameters are combined to the MLE $\hat{\Sigma}$ of the covariance matrix by matrix multiplication:

$$\hat{\Sigma} = \hat{B}^{-1}(\hat{\Lambda} + \hat{\Omega})\hat{B}^{-T}.$$

Note that in Richardson & Spirtes (2002), the matrices Λ and Ω are defined as submatrices.

Value

`Shat`	the fitted covariance matrix.
`Lhat`	matrix of the fitted precisions associated with undirected edges and vertices that do not have an arrowhead pointing at them.
`Bhat`	matrix of the fitted regression coefficients associated to the directed edges. Precisely said `Bhat` contains ones on the diagonal and the off-diagonal entry (i, j) equals the *negated* MLE of the regression coefficient for variable j in the regression of variable i on its parents. Note that this (i, j) entry in `Bhat` corresponds to a directed edge $j \to i$, and thus to a one as (j, i) entry in the adjacency matrix.
`Ohat`	matrix of the error covariances and variances of the residuals between regression equations associated with bidirected edges and vertices with an arrowhead pointing at them.
`dev`	the 'deviance' of the model.
`df`	the degrees of freedom.
`it`	the iterations.

Author(s)

Mathias Drton

References

Drton, M. & Richardson, T. S. (2003a). Iterative Conditional Fitting for Gaussian Ancestral Graph Models. Department of Statistics, University of Washington, Technical Report 437, under preparation. (Compare also `http://www.math.auc.dk/gr/gr2003/material/drton.pdf`)

Drton, M. & Richardson, T. S. (2003b). A new algorithm for maximum likelihood estimation in Gaussian graphical models for marginal independence. *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, 184–191.

Richardson, T. Spirtes, P. (2002). Ancestral Graph Markov Models. *Annals of Statistics*. 30, 4, 962–1030.

See Also

`fitCovGraph`, `icf`, `makeAG`, `fitDag`

Examples

```
## A covariance matrix
"S" <- structure(c(2.93, -1.7, 0.76, -0.06,
                  -1.7, 1.64, -0.78, 0.1,
                   0.76, -0.78, 1.66, -0.78,
                  -0.06, 0.1, -0.78, 0.81), .Dim = c(4,4),
                 .Dimnames = list(c("y", "x", "z", "u"), c("y", "x", "z", "u")))
## The following should give the same fit.
## Fit an ancestral graph y -> x <-> z <- u
fitAncestralGraph(ag1 <- makeAG(dag=DAG(x~y,z~u), bg = UG(~x*z)), S, n=100)

## Fit an ancestral graph y <-> x <-> z <-> u
fitAncestralGraph(ag2 <- makeAG(bg= UG(~y*x+x*z+z*u)), S, n=100)

## Fit the same graph with fitCovGraph
fitCovGraph(ag2, S, n=100)

## Another example for the mathematics marks data

data(marks)
S <- var(marks)
mag1 <- makeAG(bg=UG(~mechanics*vectors*algebra+algebra*analysis*statistics))
fitAncestralGraph(mag1, S, n=88)

mag2 <- makeAG(ug=UG(~mechanics*vectors+analysis*statistics),
               dag=DAG(algebra~mechanics+vectors+analysis+statistics))
fitAncestralGraph(mag2, S, n=88) # Same fit as above
```

`fitConGraph` *Fitting of Gaussian concentration graph models*

Description

Fits a concentration graph (a covariance selection model) to a sample covariance matrix, assuming a Gaussian model.

Usage

```
fitConGraph(amat, S, n, pri = FALSE, alg=2, tol = 1e-06)
```

Arguments

`amat`	a square Boolean matrix representing the adjacency matrix of the DAG
`S`	a symmetric positive definite matrix, the sample covariance matrix
`n`	an postive integer, the sample size
`pri`	a logical value. If `TRUE` a the value of the deviance at each iteration is printed.
`alg`	and integer specifying the algorithm: if `alg=1` then the covariance matrix is updated, if `alg=2` the concentration matrix is updated.
`tol`	a small positive number indicating the tolerance used in convergence tests.

Details

Algorithms for fitting Gaussian graphical models specified by undirected graphs are discussed in Speed & Kiiveri (1986). This function is based on the iterative proportional fitting algorithm described on p. 184 of Whittaker (1990).

Value

`Shat`	the fitted covariance matrix.
`dev`	the 'deviance' of the model.
`df`	the degrees of freedom.
`it`	the iterations.

Author(s)

Giovanni M. Marchetti

References

Cox, D. R. & Wermuth, N. (1996). *Multivariate dependencies.* London: Chapman & Hall.

Speed, T.P. & Kiiveri, H (1986). Gaussian Markov distributions over finite graphs. *Annals of Statistics*, 14, 138–150.

Whittaker, J. (1990). *Graphical models in applied multivariate statistics.* Chichester: Wiley.

See Also

`UG`, `fitDag`, `cliques`, `marks`

Examples

```
## A model for the sample covariance matrix of the
## mathematics marks (Whittaker, 1990)
data(marks)
S <- cov(marks) * 87 / 88
## A butterfly concentration graph
fitConGraph(UG(~ mechanics*vectors*algebra + algebra*analysis*statistics), S , n = 88)
```

`fitCovGraph` *Fitting of Gaussian covariance graph models*

Description

Fits a Gaussian covariance graph by maximum likelihood.

Usage

```
fitCovGraph(amat, S, n, alg = "icf", dual.alg = 2, start.icf = NULL, tol = 1e-06)
```

Arguments

`amat`	A symmetric Booloean matrix with dimnames representing the adjacency matrix of the graph.
`S`	A symmetric positive definite matrix with dimnames, the sample covariance matrix
`n`	A positive integer, the sample size.
`alg`	A character string, the algorithm used. If `alg="icf"` (the default) the algorithm is based on iterative conditional fitting (see Drton and Richardson, 2003). In this case the ML estimates are returned. If `alg="dual"` the algorithm is based on the dual likelihood (see Kauermann, 1996). The fitted values are not true ML estimates.
`dual.alg`	And integer equal to 1 or 2. It is used if `alg="dual"`. In this case a concentration graph model is fitted to the inverse of the sample covariance matrix, and `dual.alg` is passed to `fitConGraph` to specify the algorithm used in `fitConGraph`.
`start.icf`	A symmetric matrix used as starting value of the algorithm. If `start=NULL` the starting value is a diagonal matrix with diagonal entries equal to sample variances.
`tol`	A small positive number indicating the tolerance used in convergence tests.

Details

A covariance graph is an undirected graph in which the variables associated to two non-adjacent nodes are marginally independent. The edges of these models are represented by bidirected edges (Drton & Richardson, 2003) or by dashed lines (Cox & Wermuth, 1996).

By default, this function gives the ML estimates in the covariance graph model, by a new iterative method (Drton & Richardson, 2003). If desired then estimates from a "dual likelihood" heuristic (Kauermann, 1996; Edwards, 2000, §7.4).

Value

`Shat`	the fitted covariance matrix.
`dev`	the 'deviance' of the model.
`df`	the degrees of freedom.
`it`	the iterations.

Author(s)

Mathias Drton

References

Cox, D. R. & Wermuth, N. (1996). *Multivariate dependencies.* London: Chapman & Hall.

Drton, M. & Richardson, T. S. (2003). A new algorithm for maximum likelihood estimation in Gaussian graphical models for marginal independence. *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, 184–191.

Kauermann, G. (1996). On a dualization of graphical Gaussian models. *Scandinavian Journal of Statistics.* 23, 105–116.

See Also

fitConGraph, icf

Examples

```
## Correlations among four strategies to cope with stress for
## 72 students. Cox & Wermuth (1996), p. 73.
##  Y = cognitive avoidance
##  X = vigilance
##  V = blunting
##  U = monitoring

R <- matrix(c(
   1.00, -0.20,  0.46,  0.01,
  -0.20,  1.00,  0.00,  0.47,
   0.46,  0.00,  1.00, -0.15,
   0.01,  0.47, -0.15,  1.00), 4, 4)
nam <- c("Y", "X", "V", "U")
dimnames(R) <- list(nam, nam)

## A chordless 4-cycle covariance graph
gr <- UG(~ Y*X + X*U + U*V + V*Y)
fitCovGraph(gr, R, n=72)
fitCovGraph(gr, R, n=72, alg="dual")
```

---

fitDag *Fitting of Gaussian DAG models*

Description

Fits linear recursive regressions with independent residuals specified by a DAG.

Usage

```
fitDag(amat, S, n)
```

Arguments

amat	a square matrix with dimnames representing the adjacency matrix of the DAG
S	a symmetric positive definite matrix, the sample covariance matrix
n	an integer > 0, the sample size

Details

`fitDag` checks if the order of the nodes in adjacency matrix is the same of `S` and if not it reorders the adjacency matrix to match the order of the variables in `S`. The nodes of the adjacency matrix may form a subset of the variables in `S`.

Value

`Shat`	the fitted covariance matrix.
`Ahat`	a square matrix of the fitted regression coefficients. The entry `Ahat[i,j]` is minus the regression coefficient of variable `i` in the regression equation `j`. Thus there is a non zero partial regression coefficient `Ahat[i,j]` corresponding to each non zero value `amat[j,i]` in the adjacency matrix.
`Dhat`	a vector containing the partial variances of each variable given the parents.
`dev`	the 'deviance' of the model.
`df`	the degrees of freedom.

Author(s)

Giovanni M. Marchetti

References

Cox, D. R. & Wermuth, N. (1996). *Multivariate dependencies.* London: Chapman & Hall.

See Also

`DAG`, `swp`.

Examples

```
dag <- DAG(y ~ x+u, x ~ z, z ~ u)
"S" <- structure(c(2.93, -1.7, 0.76, -0.06,
                   -1.7, 1.64, -0.78, 0.1,
                    0.76, -0.78, 1.66, -0.78,
                    -0.06, 0.1, -0.78, 0.81), .Dim = c(4,4),
         .Dimnames = list(c("y", "x", "z", "u"), c("y", "x", "z", "u")))
fitDag(dag, S, 200)
```

`fitDagLatent` *Fitting Gaussian DAG models with one latent variable*

Description

Fits by maximum likelihood a Gaussian DAG model where one of the nodes of the graph is latent and it is marginalised over.

Usage

```
fitDagLatent(amat, Syy, n, latent, norm = 1, seed = 144,
             maxit = 9000, tol = 1e-06, pri = FALSE)
```

Arguments

`amat`	a square matrix with dimnames representing the adjacency matrix of the DAG.
`Syy`	a symmetric positive definite matrix, with dimnames, the sample covariance matrix of the observed variables. The set of the observed nodes of the graph must be a subset of the set of the names of the variables in `Syy`.
`n`	a positive integer, the sample size.
`latent`	the name of the latent variable.
`norm`	an integer, the kind of normalization of the latent variable. If `norm=1`, the latent is scaled to have unit variance. If `norm=2`, the latent is scaled to have unit partial variance given its parents.
`seed`	an integer, used by `set.seed` to specify a random starting point of the EM algorithm.
`maxit`	an integer denoting the maximum number of iterations allowed for the EM algorithm. If the convergence criterion is not satisfied within maxit iterations the algorithms stops and a warning message is returned.
`tol`	a small real value, denoting the tolerance used in testing convergence.
`pri`	logical, if `pri=TRUE` then the value of the deviance at each iteration is printed.

Details

For the EM algorithm used see Kiivery (1987).

Value

`Shat`	the fitted covariance matrix of all the variables including the latent one. The latent variable is the last. If `norm=1` then the variance of the latent variable is constrained to 1.
`Ahat`	a square matrix of the fitted regression coefficients. The entry `Ahat[i,j]` is minus the regression coefficient of variable `i` in the regression equation `j`. Thus there is a non zero partial regression coefficient `Ahat[i,j]` corresponding to each non zero value `amat[j,i]` in the adjacency matrix.
`Dhat`	a vector containing the partial variances of each variable given the parents. If `norm=2` then the partial variance of the latent variable is constrained to 1.
`dev`	the 'deviance' of the model.
`df`	the degrees of freedom of the model.
`it`	the number of EM algorithm iterations at convergence.

Author(s)

Giovanni M. Marchetti

References

Kiiveri,H. T. (1987). An incomplete data approach to the analysis of covariance structures. *Psychometrika*, 52, 4, 539–554.

Jöreskog, K.G. & Goldberger, A.S. (1975). Estimation of a model with multiple indicators and multiple causes of a single latent variable. *Journal of the American Statistical Association*, 10, 631–639.

See Also

fitDag, checkIdent

Examples

```
## data from Joreskog and Goldberger (1975)
V <- matrix(c(1,     0.36,   0.21,  0.10,  0.156, 0.158,
              0.36,  1,      0.265, 0.284, 0.192, 0.324,
              0.210, 0.265,  1,     0.176, 0.136, 0.226,
              0.1,   0.284,  0.176, 1,     0.304, 0.305,
              0.156, 0.192,  0.136, 0.304, 1,     0.344,
              0.158, 0.324,  0.226, 0.305, 0.344, 1),     6,6)
nod <- c("y1", "y2", "y3", "x1", "x2", "x3")
dimnames(V) <- list(nod,nod)
dag <- DAG(y1 ~ z, y2 ~ z, y3 ~ z, z ~ x1 + x2 + x3, x1~x2+x3, x2~x3)
fitDagLatent(dag, V, n=530, latent="z", seed=4564)
fitDagLatent(dag, V, n=530, latent="z", norm=2, seed=145)
```

`fundCycles` *Fundamental cycles*

Description

Finds the list of fundamental cycles of a connected undirected graph.

Usage

```
fundCycles(amat)
```

Arguments

`amat`	a symmetric matrix with dimnames denoting the adjacency matrix of the undirected graph. The graph must be connected, otherwise the function returns an error message.

Details

All the cycles in an UG can be obtained from combination (ring sum) of the set of fundamental cycles.

Value

a list of matrices with two columns. Every component of the list is associated to a cycle. The cycle is described by a $k \times 2$ matrix whose rows are the edges of the cycle. If there is no cycle the function returns `NULL`.

Note

This function is used by `cycleMatrix` and `isGident`.

Author(s)

Giovanni M. Marchetti

References

Thulasiraman, K. & Swamy, M.N.S. (1992). *Graphs: theory and algorithms.* New York: Wiley.

See Also

UG,findPath, cycleMatrix, isGident,bfs

Examples

```
## Three fundamental cycles
fundCycles(UG(~a*b*d + d*e + e*a*f))
```

---

ggm *The package 'ggm': summary information*

Description

This package provides functions for defining and manipulating graphs and for fitting graphical Gaussian models.

Functions

The main functions can be classified as follows.

- Functions for defining graphs (undirected, directed acyclic, ancestral graphs): UG, DAG, makeAG;
- Functions for doing graph operations (parents, boundary, cliques, connected components, fundamental cycles, d-separation): pa, bd, cliques, conComp, fundCycles, dSep;
- Function for finding covariance and concentration graphs induced by marginalization and conditioning: inducedCovGraph, inducedConGraph;
- Functions for fitting by ML Gaussian DAGs, concentration graphs, covariance graphs and ancestral graphs: fitDag, fitConGraph, fitCovGraph, fitAncestralGraph;
- Functions for testing several conditional independences: shipley.test;
- Functions for checking global identification of DAG Gaussian models with one latent variable (Stanghellini-Vicard's condition for concentration graphs, new sufficient conditions for DAGs): isGident, checkIdent;
- Functions for fitting Gaussian DAG models with one latent variable: fitDagLatent.

The package is intended as a contribution to the gR-project derscribed by Lauritzen (2002).

Authors

Giovanni M. Marchetti, Dipartimento di Statistica "G. Parenti". Università di Firenze, Italy;

Mathias Drton, Department of Statistics, University of Washington, USA.

Acknowledgements

Many thanks to Fulvia Pennoni for testing some of the functions, and to Elena Stanghellini for discussion and examples. Thanks also to Michael Perlman and Thomas Richardson.

Giovanni Marchetti has been supported by MIUR, Italy, under grant scheme PRIN 2002, and Mathias Drton has been supported by NSF grant DMS-9972008 and University of Washington RRF grant 65-3010.

References

Lauritzen, S. L. (2002). gRaphical Models in R. *R News*, 3(2)39.

`glucose` *Glucose control*

Description

Data on glucose control of diabetes patients.

Usage

```
data(glucose)
```

Format

A data frame with 68 observations on the following 8 variables.

- **Y** a numeric vector, Glucose control (glycosylated haemoglobin), values up to about 7 or 8 indicate good glucose control.
- **X** a numeric vector, a score for knowdledge about the illness.
- **Z** a numeric vector, a score for fatalistic externality (mere chance determines what occurs).
- **U** a numeric vector, a score for social externaliy (powerful others are responsible).
- **V** a numeric vector, a score for internality (the patient is him or herself responsible).
- **W** a numeric vector, duration of the illness in years.
- **A** a factor, level of education, with levels `1`: at least 13 years of formal schooling, `2`: less then 13 years.
- **B** a factor, gender with levels `1`: females, `2`: males.

Details

Data on 68 patients with fewer than 25 years of diabetes. They were collected at the University of Mainz to identify psychological and socio-economic variables possibly important for glucose control, when patients choose the appropriate dose of treatment depending on the level of blood glucose measured several times per day.

The variable of primary interest is `Y`, glucose control, measured by glycosylated haemoglobin. `X`, knowdledge about the illness, is a response of secondary interest. Variables `Z`, `U` and `V` measure patients' type of attribution, called fatalistic externality, social externality and internality. These are intermediate variables. Background variables are `W`, the duration of the illness, `A` the duration of formal schooling and `A`, gender. These are intrinsic variables.

Source

Cox & Wermuth (1996), p. 229.

References

Cox, D. R. & Wermuth, N. (1996). *Multivariate dependencies.* London: Chapman & Hall.

Examples

```
data(glucose)
## See Cox & Wermuth (1996), Figure 6.3 p. 140
coplot(Y ~ W | A, data=glucose)
```

---

icf *Iterative conditional fitting*

Description

Main algorithm for MLE fitting of Gaussian Covariance Graphs and Gaussian Ancestral models.

Usage

```
icf(bi.graph, S, start = NULL, tol = 1e-06)
icfmag(mag, S, tol = 1e-06)
```

Arguments

`bi.graph`	a symmetric matrix with dimnames representing the adjacency matrix of an undirected graph.
`mag`	a square matrix representing the adjacency matrix of an ancestral graph (for example returned by `makeAG`).
`S`	a symmetric positive definite matrix, the sample covariance matrix. The order of the variables must be the same of the order of vertices in the adjacency matrix.
`start`	a symmetric matrix used as starting value of the algorithm. If `start=NULL` the starting value is a diagonal matrix.
`tol`	A small positive number indicating the tolerance used in convergence tests.

Details

These functions are not intended to be called directly by the user.

Value

`Sigmahat`	the fitted covariance matrix.
`Bhat`	matrix of the fitted regression coefficients associated to the directed edges.
`Omegahat`	matrix of the partial covariances of the residuals between regression equations.
`iterations`	the number of iterations.

Author(s)

Mathias Drton

References

Drton, M. & Richardson, T. S. (2003a). A new algorithm for maximum likelihood estimation in Gaussian graphical models for marginal independence. *Proceedings of the Ninetheen Conference on Uncertainty in Artificial Intelligence*, 184–191.

Drton, M. & Richardson, T. S. (2003b). Iterative Conditional Fitting for Gaussian Ancestral Graph Models. Department of Statistics, University of Washington, Technical Report 437, under preparation.

See Also

`fitCovGraph`, `fitAncestralGraph`

---

isAcyclic *Graph queries*

Description

Checks if a given graph is acyclic.

Usage

```
isAcyclic(amat)
```

Arguments

`amat`	a square Boolean matrix with dimnames, the adjacency matrix of a graph.

Value

a logical value, `TRUE` if the graph is acyclic and `FALSE` otherwise.

Author(s)

Giovanni M. Marchetti

References

Aho, A.V., Hopcroft, J.E. & Ullman, J.D. (1983). *Data structures and algorithms.* Reading: Addison-Wesley.

Examples

```
## A cyclic graph
d <- matrix(0,3,3)
rownames(d) <- colnames(d) <- c("x", "y", "z")
d["x","y"] <- d["y", "z"] <- d["z", "x"] <- 1
## Test if the graph is acyclic
isAcyclic(d)
```

`isGident` *G-identifiability of an UG*

Description

Tests if an undirected graph is G-identifiable.

Usage

```
isGident(amat)
```

Arguments

`amat`	a symmetric matrix with dimnames representing the adjacency matrix of an undirected graph

Details

An undirected graph is said G-identifiable if every connected component of the complementary graph contains an odd cycle (Stanghellini and Wermuth, 2003). See also Tarantola and Vicard (2002).

Value

a logical value, `TRUE` if the graph is G-identifiable and `FALSE` if it is not.

Author(s)

Giovanni M. Marchetti

References

Stanghellini, E. & Wermuth, N. (2003). On the identification of path-analysis models with one hidden variable. Submitted and available at `http://psystat.sowi.uni-mainz.de`.

Stanghellini, E. (1997). Identification of a single-factor model using graphical Gaussian rules. *Biometrika*, 84, 241–244.

Tarantola, C. & Vicard, P. (2002). Spanning trees and identifiability of a single-factor model. *Statistical Methods & Applications*, 11, 139–152.

Vicard, P. (2000). On the identification of a single-factor model with correlated residuals. *Biometrika*, 87, 199–205.

See Also

`UG`, `cmpGraph`, `cycleMatrix`

Examples

```
## A not G-identifiable UG
G1 <- UG(~ a*b + u*v)
isGident(G1)
## G-identifiable UG
G2 <- UG(~ a + b + u*v)
isGident(G2)
## G-identifiable UG
G3 <- cmpGraph(UG(~a*b*c+x*y*z))
isGident(G3)
```

`makeAG` *Ancestral Graphs*

Description

Defines an ancestral graph from the directed, undirected and undirected components and checks if the components are compatible with an ancestral graph.

Usage

```
makeAG(dag = NULL, ug = NULL, bg = NULL)
```

Arguments

`dag`	the adjacency matrix of a directed acyclic graph specifying the arrows of the ancestral graph.
`ug`	the adjacency matrix of an undirected graph specifying the lines of the ancestral graph.
`bg`	the adjacency matrix of an undirected graph specifying the bidirected edges of the ancestral graph.

Details

An ancestral graph is a mixed graph with three types of edges: undirected, directed and bidirected edges. The following conditions must hold: (i) no undirected edge meets an arrowhead; (ii) no directed cycles; (iii) spouses cannot be ancestors. For details see Richardson & Spirtes (2002).

The function checks if, given the matrices of the undirected, directed and bidirected edges, the above three conditions are respected. If so, a resulting adjacency matrix $E = [e_{ij}$ is returned, with the following convention. If (i, j) is a directed edge, then $e_{ij} = 1$ and $e_{ji} = 0$. If (i, j) is an undirected edge, then $e_{ij} = e_{ji} = 1$. Finally, if (i, j) is a bidirected edge, then $e_{ij} = e_{ji} = 2$.

Note that the three adjacency matrices must have labels and may be defined using the functions `DAG` and `UG`.

Value

a square matrix obtained by combining the three graph components into an adjacency matrix of an ancestral graph. See the details for the coding of the adjacency matrix.

Author(s)

Giovanni M. Marchetti, Mathias Drton

References

Richardson, T. Spirtes, P. (2002). Ancestral Graph Markov Models. *Annals of Statistics.* 30, 4, 962–1030.

See Also

`UG`, `DAG`

Examples

```
## Examples from Richardson and Spirtes (2002)
## Don't run: a1 <- makeAG(dag=DAG(a~b, b~d, d~c), bg=UG(~a*c))  # Not an AG. (a2) p.969
a2 <- makeAG(dag=DAG(b ~ a, d~c), bg=UG(~a*c+c*b+b*d))           # Fig. 3 (b1) p.969
 a3 <- makeAG(ug = UG(~ a*c), dag=DAG(b ~ a, d~c), bg=UG(~ b*d)) # Fig. 3 (b2) p.969
 a5 <- makeAG(bg=UG(~alpha*beta+gamma*delta), dag=DAG(alpha~gamma,
delta~beta))  # Fig. 6 p. 973
## Another Example
 a4 <- makeAG(ug=UG(~y0*y1), dag=DAG(y4~y2, y2~y1), bg=UG(~y2*y3+y3*y4))
```

marks *Mathematics marks*

Description

Examination marks of 88 students in five subjects.

Usage

```
data(marks)
```

Format

A data frame with 88 observations on the following 5 variables.

mechanics a numeric vector, mark in Mechanics

vectors a numeric vector, mark in Vectors

algebra a numeric vector, mark in Algebra

analysis a numeric vector, mark in Analysis

statistics a numeric vector, mark in Statistics

Details

Mechanics and Vectors were closed book examinations. Algebra, Analysis and Statistics were open book examinations.

Source

Mardia, K.V., Kent, J.T. & Bibby, (1979). *Multivariate analysis.* London: Academic Press.

References

Whittaker, J. (1990). *Graphical models in applied multivariate statistics.* Chichester: Wiley.

Examples

```
data(marks)
pairs(marks)
```

---

parcor *Partial correlations*

Description

Finds the matrix of the partial correlations between pairs of variables given the rest.

Usage

```
parcor(S)
```

Arguments

`S`	a symmetric positive definite matrix, representing a covariance matrix.

Details

The algorithm computes $-\sigma^{rs}/(\sigma^{rr}\sigma^{ss})^{1/2}$ where the σ^{rs} are concentrations, i.e. elements of the inverse covariance matrix.

Value

A symmetric matrix with ones along the diagonal and in position (r, s) the partial correlation between variables r and s given all the remaining variables.

Author(s)

Giovanni M. Marchetti

References

Cox, D. R. & Wermuth, N. (1996). *Multivariate dependencies.* London: Chapman & Hall.

See Also

`var`, `cor`, `correlations`

Examples

```
### Partial correlations for the mathematics marks data
data(marks)
S <- var(marks)
parcor(S)
```

pcor *Partial correlation*

Description

Computes the partial correlation between two variables given a set of other variables.

Usage

```
pcor(u, S)
```

Arguments

`u`	a vector of integers of length > 1. The first two integers are the indices of variables the correlation of which must be computed. The rest of the vector is the conditioning set.
`S`	a symmetric positive definite matrix, a sample covariance matrix.

Value

a scalar, the partial correlation matrix between variables `u[1]` and `u[2]` given `u[-c(1,2)]`.

Author(s)

Giovanni M. Marchetti

See Also

`cor`, `parcor`, `correlations`

Examples

```
data(marks)
## The correlation between vectors and algebra given analysis and statistics
 pcor(c("vectors", "algebra", "analysis", "statistics"), var(marks))
## The same
pcor(c(2,3,4,5), var(marks))
## The correlation between vectors and algebra given statistics
 pcor(c("vectors", "algebra", "statistics"), var(marks))
## The marginal correlation between analysis and statistics
pcor(c("analysis","statistics"), var(marks))
```

`pcor.test` *Test for zero partial association*

Description

Test for conditional independence between two variables, given the other ones, assuming a multivariate normal distribution.

Usage

```
pcor.test(r, q, n)
```

Arguments

`r`	a partial correlation coefficient, computed by `pcor`.
`q`	the number of variables in the conditioning set.
`n`	integer > 0, the sample size.

Value

`tval`	The Student's t-test statistic.
`df`	The degrees of freedom
`pvalue`	The P-value, assuming a two-sided alternative.

Author(s)

Giovanni M. Marchetti

See Also

`pcor`, `shipley.test`

Examples

```
## Are 2,3 independent given 1?
data(marks)
pcor.test(pcor(c(2,3,1), var(marks)), 1, n=88)
```

`rcorr` *Random correlation matrix*

Description

Generates a random correlation matrix with the method of Marsaglia and Olkin (1984).

Usage

```
rcorr(d)
```

Arguments

`d`	an integer > 0, the order of the correlation matrix.

Details

The algorithm uses `rsphere` to generate d vectors on a sphere in d-space. If Z is a matrix with such vectors as rows, then the random correlation matrix is ZZ'.

Value

a correlation matrix of order `d`.

Author(s)

Giovanni M. Marchetti

References

Marshall, G.& Olkin, I. (1984).Generating correlation matrices. *SIAM J. Sci. Stat. Comput.*, 5, 2, 470–475.

See Also

`rsphere`

Examples

```
## A random correlation matrix of order 3
rcorr(3)
## A random correlation matrix of order 5
rcorr(5)
```

`rnormDag` *Random sample from a decomposable Gaussian model*

Description

Generates a sample from a mean centered multivariate normal distribution whose covariance matrix has a given triangular decomposition.

Usage

```
rnormDag(n, A, Delta)
```

Arguments

`n`	an integer > 0, the sample size.
`A`	a square, upper triangular matrix with ones along the diagonal. It defines, together with `Delta`, the concentration matrix (and also the covariance matrix) of the multivariate normal. The order of `A` is the number of components of the normal.
`Delta`	a numeric vector of length equal to the number of columns of `A`.

Details

The value in position (i, j) of `A` (with $i < j$) is a regression coefficient (with sign changed) in the regression of variable i on variables $i + 1, \ldots, d$.

The value in position i of `Delta` is the residual variance in the above regression.

Value

a matrix with `n` rows and `nrow(A)` columns, a sample from a multivariate normal distribution with mean zero and covariance matrix `S = solve(A) %*% diag(Delta) %*% t(solve(A))`.

Author(s)

Giovanni M. Marchetti

References

Cox, D. R. & Wermuth, N. (1996). *Multivariate dependencies*. London: Chapman & Hall.

See Also

`triDec`, `fitDag`

Examples

```
## Generate a sample of 100 observation from a multivariate normal
## The matrix of the path coefficients
A <- matrix(
c(1, -2, -3,  0, 0,  0,  0,
  0,  1,  0, -4, 0,  0,  0,
  0,  0,  1,  2, 0,  0,  0,
  0,  0,  0,  1, 1, -5,  0,
  0,  0,  0,  0, 1,  0,  3,
  0,  0,  0,  0, 0,  1, -4,
  0,  0,  0,  0, 0,  0,  1), 7, 7, byrow=TRUE)
D <- rep(1, 7)
X <- rnormDag(100, A, D)

## The true covariance matrix
solve(A) %*% diag(D) %*% t(solve(A))

## Triangular decomposition of the sample covariance matrix
triDec(cov(X))$A
```

`rsphere` *Random vectors on a sphere*

Description

Generates a sample of points uniformly distributed on the surface of a sphere in d-space.

Usage

```
rsphere(n, d)
```

Arguments

`n`	an integer, the sample size.
`d`	an integer, the dimension of the space. For example, a circle is defined in 2D-space, a sphere in 3D-space.

Details

The algorithm is based on normalizing to length 1 each d-vector of a sample from a multivariate normal $N(0, I)$.

Value

a matrix of `n` rows and `d` columns.

Author(s)

Giovanni M. Marchetti

See Also

`rnorm`, `rcorr`

Examples

```
## 100 points on circle
z <- rsphere(100,2)
plot(z)

## 100 points on a sphere
z <- rsphere(100, 3)
pairs(z)
```

`shipley.test` *Test of all independencies implied by a given DAG*

Description

Computes a simultaneous test of all independence relationships implied by a given Gaussian model defined according to a directed acyclic graph, based on the sample covariance matrix.

Usage

```
shipley.test(amat, S, n)
```

Arguments

`amat`	a square Boolean matrix, of the same dimension as `S`, representing the adjacency matrix of a DAG.
`S`	a symmetric positive definite matrix, the sample covariance matrix.
`n`	a positive integer, the sample size.

Details

The test statistic is $C = -2 \sum \ln p_j$ where p_j are the p-values of tests of conditional independence in the basis set computed by `basiSet(A)`. The p-values are independent uniform variables on $(0, 1)$ and the statistic has exactly a chi square distribution on $2k$ degrees of freedom where k is the number of elements of the basis set. Shipley (2002) calls this test Fisher's C test.

Value

`ctest`	Test statistic C.
`df`	Degrees of freedom.
`pvalue`	The P-value of the test, assuming a two-sided alternative.

Author(s)

Giovanni M. Marchetti

References

Shipley, B. (2000). A new inferential test for path models based on directed acyclic graphs. *Structural Equation Modeling*, 7(2), 206–218.

See Also

`basiSet`, `pcor.test`

Examples

```
## A decomposable model for the mathematics marks data
data(marks)
dag <- DAG(mechanics ~ vectors+algebra, vectors ~ algebra, statistics ~ algebra+analysis, analysis ~ a
shipley.test(dag, cov(marks), n=88)
```

---

swp *Sweep operator*

Description

Sweeps a covariance matrix with respect to a subset of indices.

Usage

```
swp(V, b)
```

Arguments

`V`	a symmetric positive definite matrix, the covariance matrix.
`b`	a subset of indices of the columns of `V`.

Details

The sweep operator has been introduced by Beaton (1964) as a tool for inverting symmetric matrices (see Dempster, 1969).

Value

a square matrix `U` of the same order as `V`. If `a` is the complement of `b`, then `U[a,b]` is the matrix of regression coefficients of `a` given `b` and `U[a,a]` is the corresponding covariance matrix of the residuals.

If `b` is empty the function returns `V`.

If `b` is the vector `1:nrow(V)` (or its permutation) then the function returns the opposite of the inverse of `V`.

Author(s)

Giovanni M. Marchetti

References

Beaton, A.E. (1964). *The use of special matrix operators in statistical calculus.* Ed.D. thesis, Harvard University. Reprinted as Educational Testing Service Research Bulletin 64-51. Princeton.

Dempster, A.P. (1969). *Elements of continuous multivariate analysis.* Reading: Addison-Wesley.

See Also

`fitDag`

Examples

```
## A very simple example
V <- matrix(c(10, 1, 1, 2), 2, 2)
swp(V, 2)
```

`topSort` *Topological sort*

Description

`topOrder` returns the topological order of a directed acyclic graph (parents, before children). `topSort` permutates the adjacency matrix according to the topological order.

Usage

```
topSort(amat)
topOrder(amat)
```

Arguments

`amat`	a square Boolean matrix with dimnames, representing the adjacency matrix of a directed acyclic graph.

Details

The topological order needs not to be unique. After the permutation the adjacency matrix of the graph is upper triangular. The function is a translation of the Matlab function `topological_sort` in Toolbox **BNT** written by Kevin P. Murphy.

Value

`topSort(amat)` returns a vector of integers representing the permutation of the nodes. `topSort(amat)` returns the adjacency matrix with rows and columns permutated.

Note

The order of the nodes defined by `DAG` is that of their first appearance in the model formulae (from left to right).

Author(s)

Kevin P. Murphy, Giovanni M. Marchetti

References

Aho, A.V., Hopcrtoft, J.E. & Ullman, J.D. (1983). *Data structures and algorithms.* Reading: Addison-Wesley.

Lauritzen, S. (1996). *Graphical models.* Oxford: Clarendon Press.

See Also

`DAG`, `isAcyclic`

Examples

```
## A simple example
dag <- DAG(a ~ b, c ~ a + b, d ~ c + b)
dag
topOrder(dag)
topSort(dag)
```

`transClos` *Transitive closure of a graph*

Description

Computes the transitive closure of a graph (undirected or directed acyclic).

Usage

```
transClos(amat)
```

Arguments

`amat`	a Boolean matrix with dimnames representing the adjacency matrix of a graph.

Details

The transitive closure of a directed graph with adjacency matrix A is a graph with adjacency matrix A^* such that $A^*_{i,j} = 1$ if there is a directed path from i to j. The transitive closure of an undirected graph is defined similarly (by substituting path to directed path).

Value

`A`	The adjacency matrix of the transitive closure.

Author(s)

Giovanni M. Marchetti

References

See Also

`DAG`, `UG`

Examples

```
## Closure of a DAG
d <- DAG(y ~ x, x ~ z)
transClos(d)

## Closure of an UG
g <- UG(~ x*y*z+z*u+u*v)
transClos(g)
```

`triDec` *Triangular decomposition of a covariance matrix*

Description

Decomposes a symmetric positive definite matrix with a variant of the Cholesky decomposition.

Usage

```
triDec(Sigma)
```

Arguments

`Sigma`	a symmetric positive definite matrix.

Details

Any symmetric positive definite matrix Σ can be decomposed as $\Sigma = B\Delta B^T$ where B is upper triangular with ones along the main diagonal and Δ is diagonal. If Σ is a covariance matrix, the concentration matrix is $\Sigma^{-1} = A^T\Delta^{-1}A$ where $A = B^{-1}$ is the matrix of the regression coefficients (with the sign changed) of a system of linear recursive regression equations with independent residuals. In the equations each variable i is regressed on the variables $i+1, \ldots, d$. The elements on the diagonal of Δ are the partial variances.

Value

`A`	a square upper triangular matrix of the same order as `Sigma` with ones on the diagonal.
`B`	the inverse of `A`, another triangular matrix with unit diagonal.
`Delta`	a vector containing the diagonal values of Δ.

Author(s)

Giovanni M. Marchetti

References

Cox, D. R. & Wermuth, N. (1996). *Multivariate dependencies.* London: Chapman & Hall.

See Also

`chol`

Examples

```
## Triangular decomposition of a covariance matrix
B <- matrix(c(1,  -2, 0, 1,
              0,   1, 0, 1,
              0,   0, 1, 0,
              0,   0, 0, 1), 4, 4, byrow=TRUE)
B
D <- diag(c(3, 1, 2, 1))
S <- B %*% D %*% t(B)
triDec(S)
solve(B)
```

`unmakeAG` *Ancestral Graph components*

Description

Splits the adjacency matrix of an ancestral graph into undirected and bidirected components.

Usage

```
unmakeAG(mag)
```

Arguments

`mag`	a square matrix, with dimnames, representing an ancestral graph.

Value

It is the inverse of `makeAG`. It returns the following components.

`dag`	the adjacency matrix of the directed edges.
`ug`	the adjacency matrix of the undirected edges.
`bg`	the adjacency matrix of the bidirected edges.

Author(s)

Mathias Drton

See Also

`makeAG`

Examples

```
ag <- makeAG(ug=UG(~y0*y1), dag=DAG(y4~y2, y2~y1), bg=UG(~y2*y3+y3*y4))
unmakeAG(ag)
```

Index