

Package ‘dagitty’

August 26, 2016

Type Package

Title Graphical Analysis of Structural Causal Models

Version 0.2-2

Date 2016-08-26

Author Johannes Textor, Benito van der Zander

Maintainer Johannes Textor <johannes.textor@gmx.de>

Description A port of the web-based software 'DAGitty', available at <<http://dagitty.net>>, for analyzing structural causal models (also known as directed acyclic graphs or DAGs). This package computes covariate adjustment sets for estimating causal effects, enumerates instrumental variables, derives testable implications (d-separation and vanishing tetrads), generates equivalent models, and includes a simple facility for data simulation.

VignetteBuilder knitr

Encoding UTF-8

LazyData true

License GPL-2

URL <http://www.dagitty.net>, <https://github.com/jtextor/dagitty>

BugReports <https://github.com/jtextor/dagitty/issues>

Depends R (>= 3.0.0)

Imports V8, jsonlite, boot, MASS, methods, grDevices, stats, utils, graphics

Suggests knitr, base64enc (>= 0.1-3), lavaan, testthat

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-08-26 18:58:49

R topics documented:

adjustmentSets	2
ancestorGraph	4
AncestralRelations	4
as.dagitty	5
backDoorGraph	6
canonicalize	7
coordinates	7
dagitty	8
dconnected	11
downloadGraph	12
edges	12
EquivalentModels	13
exogenousVariables	14
getExample	14
graphLayout	15
graphType	16
impliedConditionalIndependencies	16
instrumentalVariables	17
is.dagitty	18
isAdjustmentSet	18
lavaanToGraph	19
localTests	20
moralize	21
names.dagitty	22
orientPDAG	22
paths	23
plot.dagitty	24
plotLocalTestResults	24
randomDAG	25
simulateSEM	25
vanishingTetrads	27
VariableStatus	28
Index	30

adjustmentSets

Covariate Adjustment Sets

Description

Enumerates sets of covariates that (asymptotically) allow unbiased estimation of causal effects from observational data, assuming that the input causal graph is correct.

Usage

```
adjustmentSets(x, exposure = NULL, outcome = NULL, type = c("minimal",
  "canonical", "all"), effect = c("total", "direct"))
```

Arguments

x	the input graph, a DAG, MAG, PDAG, or PAG.
exposure	name(s) of the exposure variable(s). If not given (default), then the exposure variables are supposed to be defined in the graph itself.
outcome	name(s) of the outcome variable(s), also taken from the graph if not given.
type	which type of adjustment set(s) to compute. If type="minimal", then only minimal sufficient adjustment sets are returned (default). For type="all", all valid adjustment sets are returned. For type="canonical", a single adjustment set is returned that consists of all (possible) ancestors of exposures and outcomes, minus (possible) descendants of nodes on proper causal paths. This canonical adjustment set is always valid if any valid set exists at all.
effect	which effect is to be identified. If effect="total", then the total effect is to be identified, and the adjustment criterion by Perkovic et al (2015; see also van der Zander et al., 2014), an extension of Pearl's back-door criterion, is used. Otherwise, if effect="direct", then the average direct effect is to be identified, and Pearl's single-door criterion is used (Pearl, 2009). In a structural equation model (Gaussian graphical model), direct effects are simply the path coefficients.

Details

If the input graph is a MAG or PAG, then it must not contain any undirected edges (=hidden selection variables).

References

- J. Pearl (2009), *Causality: Models, Reasoning and Inference*. Cambridge University Press.
- B. van der Zander, M. Liskiewicz and J. Textor (2014), Constructing separators and adjustment sets in ancestral graphs. In *Proceedings of UAI 2014*.
- E. Perkovic, J. Textor, M. Kalisch and M. H. Maathuis (2015), A Complete Generalized Adjustment Criterion. In *Proceedings of UAI 2015*.

Examples

```
# The M-bias graph showing that adjustment for
# pre-treatment covariates is not always valid
g <- dagitty( "dag{ x -> y ; x <-> m <-> y }" )
adjustmentSets( g, "x", "y" ) # empty set
# Generate data where true effect (=path coefficient) is .5
set.seed( 123 ); d <- simulateSEM( g, .5, .5 )
confint( lm( y ~ x, d ) )["x",] # includes .5
confint( lm( y ~ x + m, d ) )["x",] # does not include .5

# Adjustment sets can also sometimes be computed for graphs in which not all
```

```
# edge directions are known
g <- dagitty("pdag { x[e] y[o] a -- {i z b}; {a z i} -> x -> y <- {z b} }")
adjustmentSets( g )
```

ancestorGraph *Ancestor Graph*

Description

Creates the induced subgraph containing only the vertices in *v*, their ancestors, and the edges between them. All other vertices and edges are discarded.

Usage

```
ancestorGraph(x, v = NULL)
```

Arguments

x the input graph, a DAG, MAG, or PDAG.
v variable names.

Details

If the input graph is a MAG or PDAG, then all *possible* ancestors will be returned (see Examples).

Examples

```
g <- dagitty("dag{ z <- x -> y }")
ancestorGraph( g, "z" )

g <- dagitty("pdag{ z -- x -> y }")
ancestorGraph( g, "y" ) # includes z
```

AncestralRelations *Ancestral Relations*

Description

Retrieve the names of all variables in a given graph that are in the specified ancestral relationship to the input variable *v*.

Usage

descendants(x, v)

ancestors(x, v)

children(x, v)

parents(x, v)

neighbours(x, v)

spouses(x, v)

adjacentNodes(x, v)

markovBlanket(x, v)

Arguments

x the input graph, of any type.

v name(s) of variable(s).

descendants(x, v) retrieves variables that are reachable from v via a directed path.

ancestors(x, v) retrieves variables from which v is reachable via a directed path.

children(x, v) finds all variables w connected to v by an edge $v \rightarrow w$.

parents(x, v) finds all variables w connected to v by an edge $w \rightarrow v$.

markovBlanket(x, v) returns x's parents, its children, and all other parents of its children. The Markov blanket always renders x independent of all other nodes in the graph.

By convention, descendants(x, v) and ancestors(x, v) include v but children(x, v) and parents(x, v) do not.

Examples

```
g <- dagitty("graph{ a <-> x -> b ; c -- x <- d }")
descendants(g, "x")
parents(g, "x")
spouses(g, "x")
```

as.dagitty

Convert to DAGitty object

Description

Converts its argument to a DAGitty object, if possible.

Usage

```
as.dagitty(x, ...)
```

Arguments

x an object.
 ... further arguments passed on to methods.

backDoorGraph	<i>Back-Door Graph</i>
---------------	------------------------

Description

Removes every first edge on a proper causal path from x. If x is a MAG or PAG, then only “visible” directed edges are removed (Zhang, 2008).

Usage

```
backDoorGraph(x)
```

Arguments

x the input graph, a DAG, MAG, PDAG, or PAG.

References

J. Zhang (2008), Causal Reasoning with Ancestral Graphs. *Journal of Machine Learning Research* 9: 1437-1474.

Examples

```
g <- dagitty( "dag { x <-> m <-> y <- x }" )
backDoorGraph( g ) # x->y edge is removed

g <- dagitty( "mag { x <-> m <-> y <- x }" )
backDoorGraph( g ) # x->y edge is not removed

g <- dagitty( "mag { x <-> m <-> y <- x <- i }" )
backDoorGraph( g ) # x->y edge is removed
```

canonicalize	<i>Canonicalize an Ancestral Graph</i>
--------------	--

Description

Takes an input ancestral graph (a graph with directed, bidirected and undirected edges) and converts it to a DAG by replacing every bidirected edge $x \leftrightarrow y$ with a substructure $x \leftarrow L \rightarrow y$, where L is a latent variable, and every undirected edge $x - y$ with a substructure $x \rightarrow S \leftarrow y$, where S is a selection variable. This function does not check whether the input is actually an ancestral graph.

Usage

```
canonicalize(x)
```

Arguments

`x` the input graph, a DAG or MAG.

Value

A list containing the following components:

- `g` The resulting graph.
- `LNames` Names of newly inserted latent variables.
- `SNames` Names of newly inserted selection variables.

Examples

```
canonicalize("mag{x<->y--z}") # introduces two new variables
```

coordinates	<i>Plot Coordinates of Variables in Graph</i>
-------------	---

Description

The DAGitty syntax allows specification of plot coordinates for each variable in a graph. This function extracts these plot coordinates from the graph description in a `dagitty` object. Note that the coordinate system is undefined, typically one needs to compute the bounding box before plotting the graph.

Usage

```
coordinates(x)
```

```
coordinates(x) <- value
```

Arguments

- `x` the input graph, of any type.
- `value` a list with components `x` and `y`, giving relative coordinates for each variable. This format is suitable for [xy.coords](#).

See Also

Function [graphLayout](#) for automatically generating layout coordinates, and function [plot.dagitty](#) for plotting graphs.

Examples

```
## Plot localization of each node in the Shrier example
plot( coordinates( getExample("Shrier") ) )

## Define a graph and set coordinates afterwards
x <- dagitty('dag{
  G <-> H <-> I <-> G
  D <- B -> C -> I <- F <- B <- A
  H <- E <- C -> G <- D
}')
coordinates( x ) <-
  list( x=c(A=1, B=2, D=3, C=3, F=3, E=4, G=5, H=5, I=5),
        y=c(A=0, B=0, D=1, C=0, F=-1, E=0, G=1, H=0, I=-1) )
plot( x )
```

 dagitty

Parse DAGitty Graph

Description

Constructs a dagitty graph object from a textual description.

Usage

```
dagitty(x, layout = FALSE)
```

Arguments

- `x` character, string describing a graphical model in dagitty syntax.
- `layout` logical, whether to automatically generate layout coordinates for each variable (see [graphLayout](#))

Details

The textual syntax for DAGitty graph is based on the dot language of the graphviz software (<http://www.graphviz.org/content/dot-language>). This is a fairly intuitive syntax – use the examples below and in the other functions to get you started. An important difference to graphviz is that the DAGitty language supports several types of graphs, which have different semantics. However, many users will mainly focus on DAGs.

A DAGitty graph description has the following form:

```
[graph type] '{' [statements] '}'
```

where [graph type] is one of 'dag', 'mag', 'pdag', or 'pag' and [statements] is a list of variables statements and edge statements, which may (optionally) be separated by semicolons. Whitespace, including newlines, has no semantic role.

Variable statements look like

```
[variable id] '[' [properties] ']'
```

For example, the statement

```
x [exposure, pos="1,0"]
```

declares a variable with ID x that is an exposure variable and has a layout position of 1,0.

The edge statement

```
x -> y
```

declares a directed edge from variable x to variable y. Explicit variable statements are not required for the variables involved in edge statements, unless attributes such as position or exposure/outcome status need to be set.

DAGs (directed acyclic graphs) can contain the following edges: \rightarrow , \leftrightarrow . Bidirected edges in DAGs are simply shorthands for substructures $\leftarrow U \rightarrow$, where U is an unobserved variable.

MAGs (maximal ancestral graphs) can contain the following edges: \rightarrow , \leftrightarrow , $--$. The bidirected and directed edges of MAGs can represent latent confounders, and the undirected edges represent latent selection variables. For details, see Richardson and Spirtes (2002).

PDAGs (partially directed acyclic graphs) can contain the following edges: \rightarrow , \leftrightarrow , $--$. The bidirected edges mean the same thing as in DAGs. The undirected edges represent edges whose direction is not known. Thus, PDAGs are used to represent equivalence classes of DAGs (see also the function [equivalenceClass](#)).

PAGs (partial ancestral graphs) are to MAGs what PDAGs are to DAGs: they represent equivalence classes of MAGs. MAGs can contain the following edges: $\textcircled{-}$, \rightarrow , $\textcircled{-}$, $--$, $\textcircled{-}$ (the @ symbols are written as circle marks in most of the literature). For details on PAGs, see Zhang et al (2008). For now, only a few DAGitty functions support PAGs (for instance, [adjustmentSets](#)).

The DAGitty parser does not perform semantic validation. That is, it will not check whether a DAG is actually acyclic, or whether all chain components in a PAG are actually chordal. This is not done because it can be computationally rather expensive.

References

Richardson, Thomas; Spirtes, Peter (2002), Ancestral graph Markov models. *The Annals of Statistics* 30(4): 962-1030.

J. Zhang (2008), Causal Reasoning with Ancestral Graphs. *Journal of Machine Learning Research* 9: 1437-1474.

B. van der Zander and M. Liskiewicz (2016), Separators and Adjustment Sets in Markov Equivalent DAGs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16)*, Phoenix, Arizona, USA.

Examples

```
# Specify a simple DAG containing one path
g <- dagitty("dag{
  a -> b ;
  b -> c ;
  d -> c
}")
# Newlines and semicolons are optional
g <- dagitty("dag{
  a -> b b -> c c -> d
}")
# Paths can be specified in one go; the semicolon below is
# optional
g <- dagitty("dag{
  a -> b ->c ; c -> d
}")
# Edges can be written in reverse notation
g <- dagitty("dag{
  a -> b -> c <- d
}")
# Spaces are optional as well
g <- dagitty("dag{a->b->c<-d}")
# Variable attributes can be set in square brackets
# Example: DAG with one exposure, one outcome, and one unobserved variable
g <- dagitty("dag{
  x -> y ; x <- z -> y
  x [exposure]
  y [outcome]
  z [unobserved]
}")
# The same graph as above
g <- dagitty("dag{x[e]y[o]z[u]x<-z->y<-x}")
# A two-factor latent variable model
g <- dagitty("dag {
  X <-> Y
  X -> a X -> b X -> c X -> d
  Y -> a Y -> b Y -> c Y -> d
}")
# Curly braces can be used to "group" variables and
# specify edges to whole groups of variables
# The same two-factor model
g <- dagitty("dag{ {X<->Y} -> {a b c d} }")
# A MAG
g <- dagitty("mag{ a -- x -> y <-> z }")
# A PDAG
```

```
g <- dagitty("pdag{ x -- y -- z }")
# A PAG
g <- dagitty("pag{ x @-@ y @-@ z }")
```

dconnected

d-Separation

Description

A set Z d -separates a path p if (1) Z contains a non-collider on p , e.g. $x \rightarrow m \rightarrow y$ with $Z=c("m")$; or (2) some collider on p is not on Z , e.g. $x \rightarrow m \leftarrow y$ with $Z=c()$.

Usage

```
dconnected(x, X, Y = list(), Z = list())
```

```
dseparated(x, X, Y = list(), Z = list())
```

Arguments

<code>x</code>	the input graph, a DAG, PDAG, or MAG.
<code>X</code>	vector of variable names.
<code>Y</code>	vector of variable names.
<code>Z</code>	vector of variable names.

`dseparated(x,X,Y,Z)` checks if all paths between X and Y are d -separated by Z .

`dconnected(x,X,Y,Z)` checks if at least one path between X and Y is not d -separated by Z .

Details

The functions also work for mixed graphs with directed, undirected, and bidirected edges. The definition of a collider in such graphs is: a node where two arrowheads collide, e.g. $x \leftarrow m \leftarrow y$ but not $x \rightarrow m \leftarrow y$.

Examples

```
dconnected( "dag{x->m->y}", "x", "y", c() ) # TRUE
dconnected( "dag{x->m->y}", "x", "y", c("m") ) # FALSE
dseparated( "dag{x->m->y}", "x", "y", c() ) # TRUE
dseparated( "dag{x->m->y}", "x", "y", c("m") ) # FALSE
```

downloadGraph	<i>Load Graph from dagitty.net</i>
---------------	------------------------------------

Description

Downloads a graph that has been built and stored online using the dagitty.net GUI. Users who store graphs online will receive a unique URL for their graph, which can be fed into this function to continue working with the graph in R.

Usage

```
downloadGraph(x = "dagitty.net/mz-Tuw9")
```

Arguments

x	dagitty model URL.
---	--------------------

edges	<i>Graph Edges</i>
-------	--------------------

Description

Extracts edge information from the input graph.

Usage

```
edges(x)
```

Arguments

x	the input graph, of any type.
---	-------------------------------

Value

a data frame with the following variables:

- v name of the start node.
- w name of the end node. For symmetric edges (bidirected and undirected), the order of start and end node is arbitrary.
- e type of edge. Can be one of "->", "<->" and "--".
- x X coordinate for a control point. If this is not NA, then the edge is drawn as an [xspline](#) through the start point, this control point, and the end point. This is especially important for cases where there is more than one edge between two variables (for instance, both a directed and a bidirected edge).
- y Y coordinate for a control point.

Examples

```
## Which kinds of edges are used in the Shrier example?
levels( edges( getExample("Shrier") )$e )
```

EquivalentModels *Generating Equivalent Models*

Description

equivalenceClass(x) generates a complete partially directed acyclic graph (CPDAG) from an input DAG x. The CPDAG represents all graphs that are Markov equivalent to x: undirected edges in the CPDAG can be oriented either way, as long as this does not create a cycle or a new v-structure (a sugraph $a \rightarrow m \leftarrow b$, where a and b are not adjacent).

Usage

```
equivalenceClass(x)

equivalentDAGs(x, n = 100)
```

Arguments

x the input graph, a DAG.
n maximal number of returned graphs.

Details

equivalentDAGs(x,n) enumerates at most n DAGs that are Markov equivalent to x.

Examples

```
# How many equivalent DAGs are there for the sports DAG example?
g <- getExample("Shrier")
length(equivalentDAGs(g))
# Plot all equivalent DAGs
par( mfrow=c(2,3) )
lapply( equivalentDAGs(g), plot )
# How many edges can be reversed without changing the equivalence class?
sum(edges(equivalenceClass(g))$e == "--")
```

exogenousVariables *Retrieve Exogenous Variables*

Description

Returns the names of all variables that have no directed arrow pointing to them. Note that this does not preclude variables connected to bidirected arrows.

Usage

```
exogenousVariables(x)
```

Arguments

x the input graph, of any type.

getExample *Get Bundled Examples*

Description

Provides access to the builtin examples of the dagitty website.

Usage

```
getExample(x)
```

Arguments

x name of the example, or part thereof. Supported values are:

- "M-bias" the M-bias graph.
- "confounding" an extended confounding triangle.
- "mediator" a small model with a mediator.
- "paths" a graph with many variables but few paths
- "Sebastiani" a small part of a genetics study (Sebastiani et al., 2005)
- "Polzer" DAG from a dentistry study (Polzer et al., 2012)
- "Schipf" DAG from a study on diabetes (Schipf et al., 2010)
- "Shrier" DAG from a classic sports medicine example (Shrier & Platt, 2008)
- "Thoemmes" DAG with unobserved variables (communicated by Felix Thoemmes, 2013).
- "Kampen" DAG from a psychiatry study (van Kampen, 2014)

References

Sabine Schipf, Robin Haring, Nele Friedrich, Matthias Nauck, Katharina Lau, Dietrich Alte, Andreas Stang, Henry Voelzke, and Henri Wallaschofski (2011), Low total testosterone is associated with increased risk of incident type 2 diabetes mellitus in men: Results from the study of health in pomerania (SHIP). *The Aging Male* **14**(3):168–75.

Paola Sebastiani, Marco F. Ramoni, Vikki Nolan, Clinton T. Baldwin, and Martin H. Steinberg (2005), Genetic dissection and prognostic modeling of overt stroke in sickle cell anemia. *Nature Genetics*, **37**:435–440.

Ian Shrier and Robert W. Platt (2008), Reducing bias through directed acyclic graphs. *BMC Medical Research Methodology*, **8**(70).

Ines Polzer, Christian Schwahn, Henry Voelzke, Torsten Mundt, and Reiner Biffar (2012), The association of tooth loss with all-cause and circulatory mortality. Is there a benefit of replaced teeth? A systematic review and meta-analysis. *Clinical Oral Investigations*, **16**(2):333–351.

Dirk van Kampen (2014), The SSQ model of schizophrenic prodromal unfolding revised: An analysis of its causal chains based on the language of directed graphs. *European Psychiatry*, **29**(7):437–48.

Examples

```
g <- getExample("Shrier")
plot(g)
```

graphLayout

Generate Graph Layout

Description

This function generates plot coordinates for each variable in a graph that does not have them already. To this end, the well-known “Spring” layout algorithm is used. Note that this is a stochastic algorithm, so the generated layout will be different every time (which also means that you can try several times until you find a decent layout).

Usage

```
graphLayout(x, method = "spring")
```

Arguments

x the input graph, of any type.
method the layout method; currently, only "spring" is supported.

Value

the same graph as x but with layout coordinates added.

Examples

```
## Generate a layout for the M-bias graph and plot it
plot( graphLayout( dagitty( "dag { X <- U1 -> M <- U2 -> Y } " ) ) )
```

graphType	<i>Get Graph Type</i>
-----------	-----------------------

Description

Get Graph Type

Usage

```
graphType(x)
```

Arguments

x the input graph.

Examples

```
graphType( "mag{ x<-> y }" ) == "mag"
```

impliedConditionalIndependencies	<i>List Implied Conditional Independencies</i>
----------------------------------	--

Description

Generates a list of conditional independence statements that must hold in every probability distribution compatible with the given model.

Usage

```
impliedConditionalIndependencies(x, type = "missing.edge",
  max.results = Inf)
```


Arguments

x	the input graph, a DAG, MAG, or PDAG.
type	can be one of "missing.edge" or "basis.set". With the former, one testable implication is returned per missing edge of the graph. With the latter, one testable implication is returned per vertex of the graph that has non-descendants other than its parents. Basis sets can be smaller, but they involve higher-dimensional independencies, whereas missing edge sets involve only bivariate independencies.
max.results	integer. The listing of conditional independencies is stopped once this many results have been found. Use Inf to generate them all. This applies only when type="missing.edge".

Examples

```
g <- dagitty( "dag{ x -> m -> y }" )
impliedConditionalIndependencies( g ) # one
latents( g ) <- c("m")
impliedConditionalIndependencies( g ) # none
```

instrumentalVariables *Find Instrumental Variables*

Description

Generates a list of instrumental variables that can be used to infer the total effect of an exposure on an outcome in the presence of latent confounding, under linearity assumptions.

Usage

```
instrumentalVariables(x, exposure = NULL, outcome = NULL)
```

Arguments

x	the input graph, a DAG.
exposure	name of the exposure variable. If not given (default), then the exposure variable is supposed to be defined in the graph itself. Only a single exposure variable and a single outcome variable supported.
outcome	name of the outcome variable, also taken from the graph if not given. Only a single outcome variable is supported.

References

B. van der Zander, J. Textor and M. Liskiewicz (2015), Efficiently Finding Conditional Instruments for Causal Inference. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pp. 3243-3249. AAAI Press, 2015.

Examples

```
# The classic IV model
instrumentalVariables( "dag{ i->x->y; x<->y }", "x", "y" )
# A conditional instrumental variable
instrumentalVariables( "dag{ i->x->y; x<->y ; y<-z->i }", "x", "y" )
```

is.dagitty	<i>Test for Graph Class</i>
------------	-----------------------------

Description

A function to check whether an object has class dagitty.

Usage

```
is.dagitty(x)
```

Arguments

x	object to be tested.
---	----------------------

isAdjustmentSet	<i>Adjustment Criterion</i>
-----------------	-----------------------------

Description

Test whether a set fulfills the adjustment criterion, that means, it removes all confounding bias when estimating a **total** effect. This is an extension of Pearl's Back-door criterion (Shpitser et al, 2010; van der Zander et al, 2014; Perkovic et al, 2015) which is complete in the sense that either a set fulfills this criterion, or it does not remove all confounding bias.

Usage

```
isAdjustmentSet(x, Z, exposure = NULL, outcome = NULL)
```

Arguments

x	the input graph, a DAG, MAG, PDAG, or PAG.
Z	vector of variable names.
exposure	name(s) of the exposure variable(s). If not given (default), then the exposure variables are supposed to be defined in the graph itself.
outcome	name(s) of the outcome variable(s), also taken from the graph if not given.

Details

If the input graph is a MAG or PAG, then it must not contain any undirected edges (=hidden selection variables).

References

E. Perkovic, J. Textor, M. Kalisch and M. H. Maathuis (2015), A Complete Generalized Adjustment Criterion. In *Proceedings of UAI 2015*.

I. Shpitser, T. VanderWeele and J. M. Robins (2010), On the validity of covariate adjustment for estimating causal effects. In *Proceedings of UAI 2010*.

lavaanToGraph	<i>Convert Lavaan Model to DAGitty Graph</i>
---------------	--

Description

The lavaan package is a popular package for structural equation modeling. To provide interoperability with lavaan, this function converts models specified in lavaan syntax to dagitty graphs.

Usage

```
lavaanToGraph(x, ...)
```

Arguments

x	data frame, lavaan parameter table such as returned by lavaanify .
...	Not used.

Examples

```
if( require(lavaan) ){
  mdl <- lavaanify("
  X ~ C1 + C3
  M ~ X + C3
  Y ~ X + M + C3 + C5
  C1 ~ C2
  C3 ~ C2 + C4
  C5 ~ C4
  C1 ~~ C2 \n C1 ~~ C3 \n C1 ~~ C4 \n C1 ~~ C5
  C2 ~~ C3 \n C2 ~~ C4 \n C2 ~~ C5
  C3 ~~ C4 \n C3 ~~ C5", fixed.x=FALSE)
  plot( graphLayout( lavaanToGraph( mdl ) ) )
}
```

localTests	<i>Test Graph against Data</i>
------------	--------------------------------

Description

Derives testable implications from the given graphical model and tests them against the given dataset.

Usage

```
localTests(x, data = NULL, type = c("cis", "cis.loess", "tetrads",
  "tetrads.within", "tetrads.between", "tetrads.epistemic"), tests = NULL,
  sample.cov = NULL, sample.nobs = NULL, conf.level = 0.95, R = NULL,
  loess.pars = NULL)
```

Arguments

x	the input graph, a DAG, MAG, or PDAG.
data	matrix or data frame containing the data.
type	character indicating which kind of local test to perform. Supported values are "cis" (linear conditional independence), "cis.loess" (conditional independence using loess regression), "tetrads" and "tetrads.type", where "type" is one of the items of the tetrad typology, e.g. "tetrads.within" (see vanishingTetrads). Tetrad testing is only implemented for DAGs.
tests	optional list of the precise tests to perform. If not given, the list of tests is automatically derived from the input graph. Can be used to restrict testing to only a certain subset of tests (for instance, to test only those conditional independencies for which the conditioning set is of a reasonably low dimension, such as shown in the example).
sample.cov	the sample covariance matrix; ignored if data is supplied. Either data or sample.cov and sample.nobs must be supplied.
sample.nobs	number of observations; ignored if data is supplied.
conf.level	determines the size of confidence intervals for test statistics.
R	how many bootstrap replicates for estimating confidence intervals. If NULL, then confidence intervals are based on normal approximation. For tetrads, the normal approximation is only valid in large samples even if the data are normally distributed.
loess.pars	list of parameter to be passed on to loess (for type="cis.loess"), for example the smoothing range.

Details

Tetrad implications can only be derived if a Gaussian model (i.e., a linear structural equation model) is postulated. Conditional independence implications (CI) do not require this assumption. However, both Tetrad and CI implications are tested parametrically: for Tetrads, Wishart's confidence interval formula is used, whereas for CIs, a Z test of zero conditional covariance (if the covariance matrix is given) or a test of residual independence after linear regression (if the raw data is given) is performed. Both tetrad and CI tests also support bootstrapping instead of estimating parametric confidence intervals.

Examples

```
# Simulate full mediation model with measurement error of M1
set.seed(123)
d <- simulateSEM("dag{X->{U1 M2}->Y U1->M1}",.6,.6)

# Postulate and test full mediation model without measurement error
plotLocalTestResults(localTests( "dag{ X -> {M1 M2} -> Y }", d, "cis" ))

# Simulate data from example SEM
g <- getExample("Polzer")
d <- simulateSEM(g,.1,.1)

# Compute independencies with at most 3 conditioning variables
imp <- Filter(function(x) length(x$Z)<4, impliedConditionalIndependencies(g))
plotLocalTestResults(localTests( g, d, "cis.loess", R=100, tests=imp, loess.pars=list(span=0.6) ))
```

moralize

Moral Graph

Description

Graph obtained from x by (1) “marrying” (inserting an undirected edge between) all nodes that have common children, and then replacing all edges by undirected edges. If x contains bidirected edges, then all sets of nodes connected by a path containing only bidirected edges are treated like a single node (see Examples).

Usage

```
moralize(x)
```

Arguments

x the input graph, a DAG, MAG, or PDAG.

Examples

```
# returns a complete graph
moralize( "dag{ x->m<-y }" )
# also returns a complete graph
moralize( "dag{ x -> m1 <-> m2 <-> m3 <-> m4 <- y }" )
```

names.dagitty	<i>Names of Variables in Graph</i>
---------------	------------------------------------

Description

Extracts the variable names from an input graph. Useful for iterating over all variables.

Usage

```
## S3 method for class 'dagitty'
names(x)
```

Arguments

x the input graph, of any type.

Examples

```
## A "DAG" with Romanian and Swedish variable names. These can be
## input using quotes to overcome the limitations on unquoted identifiers.
g <- dagitty( 'digraph {
  "cora ao" [pos="0.297,0.502"]
  "h arta" [pos="0.482,0.387"]
  "cora ao" -> "h arta"
}' )
names( g )
```

orientPDAG	<i>Orient Edges in PDAG.</i>
------------	------------------------------

Description

Orients as many edges as possible in a partially directed acyclic graph (PDAG) by converting induced subgraphs $X \rightarrow Y - Z$ to $X \rightarrow Y \rightarrow Z$.

Usage

```
orientPDAG(x)
```

Arguments

x the input graph, a PDAG.

Examples

```
orientPDAG( "pdag { x -> y -- z }" )
```

paths *Show Paths*

Description

Returns a list with two components: path gives the actual paths, and open shows whether each path is open (d-connected) or closed (d-separated).

Usage

```
paths(x, from = exposures(x), to = outcomes(x), Z = list(), limit = 100,
      directed = FALSE)
```

Arguments

x the input graph, a DAG, PDAG, or MAG.

from name(s) of first variable(s).

to name(s) of last variable(s).

Z names of variables to condition on for determining open paths.

limit maximum amount of paths to show. In general, the number of paths grows exponentially with the number of variables in the graph, such that path inspection is not useful except for the most simple models.

directed logical; should only directed (i.e., causal) paths be shown?

Examples

```
sum( paths(backDoorGraph(getExample("Shrier")))$open ) # Any open Back-Door paths?
```

plot.dagitty *Plot Graph*

Description

A simple plot method to quickly visualize a graph. This is intended mainly for simple visualization purposes and not as a full-fledged graph drawing function.

Usage

```
## S3 method for class 'dagitty'
plot(x, ...)
```

Arguments

x the input graph, a DAG, MAG, or PDAG.
 ... not used.

plotLocalTestResults *Plot Results of Local Tests*

Description

Generates a summary plot of the results of local tests (see [localTests](#)). For each test, a test statistic and the confidence interval are shown.

Usage

```
plotLocalTestResults(x, xlab = "test statistic (95% CI)", xlim = range(x[,
  c(ncol(x) - 1, ncol(x))]), sort.by.statistic = TRUE, n = Inf,
  axis.pars = list(las = 1), ...)
```

Arguments

x data frame; results of the local tests as returned by [localTests](#).
 xlab X axis label.
 xlim numerical vector with 2 elements; range of X axis.
 sort.by.statistic logical. Sort the rows of x by the absolute value of the test statistic before plotting.
 n plot only the n tests for which the absolute value of the test statistics diverges most from 0.
 axis.pars arguments to be passed on to [axis](#) when generating the Y axis for the plot.
 ... further arguments to be passed on to [plot](#).

Examples

```
d <- simulateSEM("dag{X->{U1 M2}->Y U1->M1}", .6, .6)
par(mar=c(2,8,1,1)) # so we can see the test names
plotLocalTestResults(localTests( "dag{ X -> {M1 M2} -> Y }", d, "cis" ))
```

randomDAG

Generate DAG at Random

Description

Generates a random DAG with N variables called x_1, \dots, x_N . For each pair of variables x_i, x_j with $i < j$, an edge $i \rightarrow j$ will be present with probability p .

Usage

```
randomDAG(N, p)
```

Arguments

N desired number of variables.
 p connectivity parameter, a number between 0 and 1.

simulateSEM

Simulate Data from Structural Equation Model

Description

Interprets the input graph as a structural equation model, generates random path coefficients, and simulates data from the model. This is a very bare-bones function and probably not very useful except for quick validation purposes (e.g. checking that an implied vanishing tetrad truly vanishes in simulated data). For more elaborate simulation studies, please use the lavaan package or similar facilities in other packages.

Usage

```
simulateSEM(x, b.default = NULL, b.lower = -0.6, b.upper = 0.6, eps = 1,
  N = 500, standardized = TRUE)
```

Arguments

<code>x</code>	the input graph, a DAG (which may contain bidirected edges).
<code>b.default</code>	default path coefficient applied to arrows for which no coefficient is defined in the model syntax.
<code>b.lower</code>	lower bound for random path coefficients, applied if <code>b.default=NULL</code> .
<code>b.upper</code>	upper bound for path coefficients.
<code>eps</code>	residual variance (only meaningful if <code>standardized=FALSE</code>).
<code>N</code>	number of samples to generate.
<code>standardized</code>	whether a standardized output is desired (all variables have variance 1). If <code>standardized=TRUE</code> , all path coefficients are interpreted as standardized coefficients. But not all standardized coefficients are compatible with all graph structures. For instance, the graph structure <code>z <- x -> y -> z</code> is incompatible with standardized coefficients of 0.9, since this would imply that the variance of <code>z</code> must be larger than 1. For large graphs with many parallel paths, it can be very difficult to find coefficients that work.

Details

Data are generated in the following manner. Each directed arrow is assigned a path coefficient that can be given using the attribute "beta" in the model syntax (see the examples). All coefficients not set in this manner are set to the `b.default` argument, or if that is not given, are chosen uniformly at random from the interval given by `b.lower` and `b.upper` (inclusive; set both parameters to the same value for constant path coefficients). Each bidirected arrow `a <-> b` is replaced by a substructure `a <- L -> b`, where `L` is an exogenous latent variable. Path coefficients on such substructures are set to \sqrt{x} , where `x` is again chosen at random from the given interval; if `x` is negative, one path coefficient is set to $-\sqrt{x}$ and the other to \sqrt{x} . All residual variances are set to `eps`.

Value

Returns a data frame containing `N` values for each variable in `x`.

Examples

```
## Simulate data with pre-defined path coefficients of -.6
g <- dagitty('dag{z -> x [beta=-.6] x <- y [beta=-.6] }')
x <- simulateSEM( g )
cov(x)
```

vanishingTetrads *List Implied Vanishing Tetrads*

Description

Interpret the given graph as a structural equation model and list all the vanishing tetrads that it implies.

Usage

```
vanishingTetrads(x, type = NA)
```

Arguments

x the input graph, a DAG.

type restrict output to one level of Kenny's tetrad typology. Possible values are "within" (homogeneity within constructs; all four variables have the same parents), "between" (homogeneity between constructs; two pairs of variables each sharing one parent) and "epistemic" (consistency of epistemic correlations; three variables have the same parent). By default, all tetrads are listed.

Value

a data frame with four columns, where each row of the form i,j,k,l means that the tetrad $\text{Cov}(i,j)\text{Cov}(k,l) - \text{Cov}(i,k)\text{Cov}(j,l)$ vanishes (is equal to 0) according to the model.

References

Kenny, D. A. (1979), *Correlation and Causality*. Wiley, New York.

Examples

```
# Specify two-factor model with 4 indicators each
g <- dagitty("dag{{x1 x2 x3 x4} <- x <-> y -> {y1 y2 y3 y4}}")
latents(g) <- c("x","y")

# Check how many tetrads are implied
nrow(vanishingTetrads(g))
# Check how these distribute across the typology
nrow(vanishingTetrads(g,"within"))
nrow(vanishingTetrads(g,"between"))
nrow(vanishingTetrads(g,"epistemic"))
```

VariableStatus

Variable Statuses

Description

Get or set variables with a given status in a graph. Variables in dagitty graphs can have one of several statuses. Variables with status *exposure* and *outcome* are important when determining causal effects via the functions [adjustmentSets](#) and [instrumentalVariables](#). Variables with status *latent* are assumed to be unobserved variables or latent constructs, which is respected when deriving testable implications of a graph via the functions [impliedConditionalIndependencies](#) or [vanishingTetrads](#).

Usage

```
exposures(x)
exposures(x) <- value
outcomes(x)
outcomes(x) <- value
latents(x)
latents(x) <- value
adjustedNodes(x)
adjustedNodes(x) <- value
setVariableStatus(x, status, value)
```

Arguments

x	the input graph, of any type.
value	character vector; names of variables to receive the given status.
status	character, one of "exposure", "outcome" or "latent".

Details

`setVariableStatus` first removes the given status from all variables in the graph that had it, and then sets it on the given variables. For instance, if `status="exposure"` and `value="X"` are given, then X will be the only exposure in the resulting graph.

Examples

```
g <- dagitty("dag{ x<->m<->y<-x }") # m-bias graph
exposures(g) <- "x"
outcomes(g) <- "y"
adjustmentSets(g)
```

Index

adjacentNodes (AncestralRelations), 4
adjustedNodes (VariableStatus), 28
adjustedNodes<- (VariableStatus), 28
adjustmentSets, 2, 9, 28
ancestorGraph, 4
ancestors (AncestralRelations), 4
AncestralRelations, 4
as.dagitty, 5
axis, 24

backDoorGraph, 6

canonicalize, 7
children (AncestralRelations), 4
coordinates, 7
coordinates<- (coordinates), 7

dagitty, 8
dconnected, 11
descendants (AncestralRelations), 4
downloadGraph, 12
dseparated (dconnected), 11

edges, 12
equivalenceClass, 9
equivalenceClass (EquivalentModels), 13
equivalentDAGs (EquivalentModels), 13
EquivalentModels, 13
exogenousVariables, 14
exposures (VariableStatus), 28
exposures<- (VariableStatus), 28

getExample, 14
graphLayout, 8, 15
graphType, 16

impliedConditionalIndependencies, 16, 28

instrumentalVariables, 17, 28
is.dagitty, 18
isAdjustmentSet, 18

latents (VariableStatus), 28
latents<- (VariableStatus), 28
lavaanify, 19
lavaanToGraph, 19
localTests, 20, 24
loess, 20

markovBlanket (AncestralRelations), 4
moralize, 21

names.dagitty, 22
neighbours (AncestralRelations), 4

orientPDAG, 22
outcomes (VariableStatus), 28
outcomes<- (VariableStatus), 28

parents (AncestralRelations), 4
paths, 23
plot, 24
plot.dagitty, 8, 24
plotLocalTestResults, 24

randomDAG, 25

setVariableStatus (VariableStatus), 28
simulateSEM, 25
spouses (AncestralRelations), 4

vanishingTetrads, 20, 27, 28
VariableStatus, 28

xspline, 12
xy.coords, 8