Instructions on using the ggm (“Graphical Gaussian Models”) library

Topics

1. Inputting a DAG
2. dSep()
3. Shipley.test()
4. basiSet()

Inputting a DAG (directed acyclic graph)

The R function is: DAG()

**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.

Consider this simple causal chain model: X->Y->Z. To input this model as a DAG you would specify:

My.Dag<-DAG(Y~X, Z~Y) Note that this uses the same syntax as when specifying linear models in R. For each child node (dependent variable, endogenous variable) you specify

Dependent ~ parent variable1 + parent variable2 etc.

Here are some examples :

**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!

## Not 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)

The output of DAG() is 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.

Example using DAG 3.1 (p.72):

> my.dag<-DAG(B~A,C~B,D~B,E~C+D)

> my.dag

 B A C D E

B 0 0 1 1 0

A 1 0 0 0 0

C 0 0 0 0 1

D 0 0 0 0 1

E 0 0 0 0 0

dSep() function

**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.  |

**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)

Example using DAG 3.1 (p.72):

> my.dag<-DAG(B~A,C~B,D~B,E~C+D)

> dSep(my.dag,first="A",second="C",cond="B")

[1] TRUE

> dSep(my.dag,first="A",second="C",cond=NULL)

[1] FALSE

> dSep(my.dag,first="A",second="C",cond="D")

[1] FALSE

> dSep(my.dag,first="B",second="E",cond=c("C","D"))

[1] TRUE

shipley.test() function

**NOTE: I have written a modified version (shipley.test2()) that also outputs the individual tests of each element in the basis set.**

**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 ∑ \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. |

Example using DAG 3.1 (p.72):

# first enter the DAG…

> my.dag<-DAG(B~A,C~B,D~B,E~C+D)

# Now, generate 100 observations from this DAG…

> my.dat<-gen.path.model.3.1(100)

# You now have to enter theDAG (called “amat”, the covariance matrix of the observed data (S),

# and the sample size (n)…

> shipley.test(amat=my.dag,S=cov(my.dat),n=100)

$ctest

[1] 14.37551

$df

[1] 10

$pvalue

[1] 0.1565421

Here is what happens if you enter a DAG that did not actually generate your data:

> shipley.test(amat=DAG(B~A,C~B,D~B,E~B),S=cov(my.dat),n=100)

$ctest

[1] 85.40734

$df

[1] 12

$pvalue

[1] 3.800293e-13

basiSet() function

**(useful if you must apply the d-sep test manually because the data are not multivariate normal and linear)**

**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).

Example using DAG 3.1 (p.72):

> basiSet(amat=DAG(B~A,C~B,D~B,E~C+D))

[[1]]

[1] "A" "D" "B" 🡨 means A is d-sep D given B

[[2]]

[1] "A" "C" "B"

[[3]]

[1] "A" "E" "D" "C"

[[4]]

[1] "B" "E" "A" "D" "C" 🡨 means B is d-sep E given {A,D,C}

[[5]]

[1] "D" "C" "B"