Package 'gRbase'

August 7, 2018

Version 1.8-3.4

Title A Package for Graphical Modelling in R

Author Søren Højsgaard <sorenh@math.aau.dk>

Maintainer Søren Højsgaard <sorenh@math.aau.dk>

Description The 'gRbase' package provides general features
which are used by other graphical modelling packages, in particular
by the packages 'gRain', 'gRim' and 'gRc'.
'gRbase' contains several data sets relevant for use in connection with
graphical models. Almost all data sets used in the book Graphical
Models with R (2012) are contained in 'gRbase'.
'gRbase' implements several graph algorithms (based mainly on
representing graphs as adjacency matrices - either in the form
of a standard matrix or a sparse matrix). Some graph
algorithms are:
(i) maximum cardinality search (for marked and unmarked graphs).
(ii) moralize.
(iii) triangulate.
(iv) junction tree.
'gRbase' facilitates array operations,
'gRbase' implements functions for testing for conditional independence.
'gRbase' illustrates how hierarchical log-linear models may be
implemented and describes concept of graphical meta
data. These features, however, are not maintained anymore and
remains in 'gRbase' only because there exists a paper describing
these facilities: A Common Platform for Graphical Models in R:
The 'gRbase' Package, Journal of Statistical Software, Vol 14,
No 17, 2005.
NOTICE Proper functionality of 'gRbase' requires that the packages graph,
'Rgraphviz' and 'RBGL' are installed from 'bioconductor'; for
installation instructions please refer to the web page given below.

License GPL (>= 2)

URL http://people.math.aau.dk/~sorenh/software/gR/

ByteCompile Yes

Encoding UTF-8

VignetteBuilder knitr

Depends R (>= 3.0.2), methods

Imports graph, igraph, magrittr, Matrix, RBGL, Rcpp (>= 0.11.1)

Suggests Rgraphviz, microbenchmark, knitr

LinkingTo Rcpp (>= 0.11.1), RcppArmadillo, RcppEigen

RoxygenNote 5.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-08-07 21:30:06 UTC

R topics documented:

array-algebra . . . 3
array-create . . . 4
array-dimnames . . . 6
array-distribution . . . 7
array-normalize . . . 8
array-operations . . . 9
array-properties . . . 11
array-slice . . . 12
ashtrees . . . 14
BodyFat . . . 15
breastcancer . . . 16
cad . . . 17
carcass . . . 18
chestSim . . . 19
combnPrim . . . 20
compareModels . . . 21
cov2pcor . . . 22
dietox . . . 22
dumping . . . 23
graph-cliques . . . 24
graph-coerce . . . 25
graph-create . . . 26
graph-edgeList . . . 27
graph-gcproperties . . . 28
graph-iplot . . . 30
graph-is . . . 31
graph-mcs . . . 33
graph-minimaltriang . . . 35
graph-moralize . . . 37
graph-mpd . . . 38
graph-query . . . 39
graph-randomdag . . . 40
graph-rip . . . 41

graph-toposort 43
graph-triang 44
graph-triangulate 46
graph-vpar 48
graph-xxx2yyy 50
gRbase 52
gRbase-generics 53
gRbase-utilities 54
lizard 55
mathmark 56
mildew 56
milkcomp 57
Nutrimouse 58
old-array-operations 63
old-parray 64
rats 65
reinis 66
Setoperations 67
simulateArray 68
wine 69

Index **71**

`array-algebra` *Array algebra*

Description

Addition, subtraction etc. of arrays

Usage

```
a1 %a+% a2

a1 %a-% a2

a1 %a*% a2

a1 %a/% a2

a1 %a/0% a2

ar_add(a1, a2)

ar_subt(a1, a2)

ar_mult(a1, a2)
```

```
ar_div(a1, a2)

ar_div0(a1, a2)

ar_sum(...)

ar_prod(...)
```

Arguments

`a, a1, a2, ...`	Arrays (with named dimnames)

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
hec <- HairEyeColor
a1 <- ar_marg(hec, c("Hair", "Eye"))
a2 <- ar_marg(hec, c("Hair", "Sex"))
a3 <- ar_marg(hec, c("Eye", "Sex"))

## Binary operations
a1 %a+% a2
a1 %a-% a2
a1 %a*% a2
a1 %a/% a2

ar_sum(a1, a2, a3)
ar_prod(a1, a2, a3)
```

---

array-create *Create multidimensional arrays*

Description

Alternative ways of creating arrays

Convert dataframe to contingency table

Usage

```
newar(names, levels, values, normalize = "none", smooth = 0)

ar_new(names, levels, values, normalize = "none", smooth = 0)

df2xtabs(indata, names = NULL, normalize = "none", smooth = 0)
```

Arguments

names	Names of variables defining table; a character vector or a right hand sided formula.
levels	1) a list with specification of the levels of the factors in names or 2) a vector with number of levels of the factors in names. See 'examples' below.
values	values to go into the parray
normalize	Either "none", "first" or "all". Should result be normalized, see 'Details' below.
smooth	Should values be smoothed, see 'Details' below.
indata	A dataframe.

Details

If normalize="first" then for each configuration of all other variables than the first, the probabilities are normalized to sum to one. Thus f(a,b,c) becomes a conditional probability table of the form p(a|b,c). If normalize="all" then the sum over all entries of f(a,b,c) is one.

If smooth is positive then smooth is added to values before normalization takes place.

Value

An array.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

ar_perm, ar_add, ar_prod etc.

Examples

```
universe <- list(gender=c('male','female'),
                 answer=c('yes','no'),
                 rain=c('yes','no'))
t1 <- ar_new(c("gender","answer"), levels=universe, values=1:4)
t1
t2 <- ar_new(~gender:answer, levels=universe, values=1:4)
t2
t3 <- ar_new(~gender:answer, c(2,2), values=1:4)
t3

## Extract arrays from dataframe (much like xtabs() but with more flexibility)
data(cad1)
df2xtabs(cad1, ~Sex:AngPec:AMI)
df2xtabs(cad1, c("Sex","AngPec","AMI"))
df2xtabs(cad1, c(1,2,3))

## Extract arrays from dataframe (much like xtabs() but with more flexibility)
```

```
data(cad1)
df2xtabs(cad1, ~Sex:AngPec:AMI)
df2xtabs(cad1, c("Sex","AngPec","AMI"))
df2xtabs(cad1, c(1,2,3))
```

`array-dimnames` *Check compatibility of dimnames*

Description

Check that the intersection of dimnames of two arrays are identical.

Usage

```
dimnames_match(a1, a2)
```

Arguments

`a1, a2`	Arrays with named dimnames.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

`is.named.array`

Examples

```
ar1 = ar_new(c("a", "b"), levels=c(2, 3))
ar2 = ar_new(c("c", "a"), levels=c(2, 2))
ar1
ar2
## dimension a has levels a1,a2 in both ar1 and ar2.
# Hence we have a match.
dimnames_match(ar1, ar2)

ar1 = ar_new(c("a", "b"), levels=c(2, 3))
ar2 = ar_new(c("c", "a"), levels=c(2, 3))
ar1
ar2
## dimension a has levels a1,a2 in ar1 and levels a1,a2,a3 in ar2.
# Hence we do not have a match.
dimnames_match(ar1, ar2)

ar2 = ar_new(c("c", "a"), levels=list(c=c("c1", "c2"), a=c("a2", "a1")))
ar2
## dimension a has levels a1,a2 in ar1 and levels a2,a1 in ar2.
```

```
# Hence we do not have a match.
dimnames_match(ar1, ar2)
```

`array-distribution` *Marginalize and condition in multidimensional array.*

Description

Marginalize and condition in a multidimensional array which is assumed to represent a discrete multivariate distribution.

Usage

```
ar_dist(tab, marg = NULL, cond = NULL, normalize = TRUE)
```

Arguments

`tab`	Multidimensional array with dimnames.
`marg`	A specification of the desired margin; a character vector, a numeric vector or a right hand sided formula.
`cond`	A specification of what is conditioned on. Can take two forms: Form one is a a character vector, a numeric vector or a right hand sided formula. Form two is as a simple slice of the array, which is a list of the form var1=value1, var2=value2 etc.
`normalize`	Should the result be normalized to sum to 1.

Value

A multidimensional array.

Note

`ar_dist` is a recent addition and its functionality (and name) may change. `ar_dist` is based on calling `ar_marg` and `ar_slice`.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

`ar_new`, `ar_marg`, `ar_slice` etc.

Examples

```
hec <- HairEyeColor

is.named.array( hec )
## We need dimnames, and names on the dimnames

## Marginalize:
ar_dist(hec, marg= ~Hair + Eye)
ar_dist(hec, marg= ~Hair:Eye)
ar_dist(hec, marg= c("Hair", "Eye"))
ar_dist(hec, marg= 1:2)

ar_dist(hec, marg= ~Hair + Eye, normalize=FALSE)

## Condition
ar_dist(hec, cond= ~Sex + Hair)
ar_dist(hec, cond= ~Sex:Hair)
ar_dist(hec, cond= c("Sex", "Hair"))
ar_dist(hec, cond= c(3,1))

ar_dist(hec, cond= list(Hair="Black"))
ar_dist(hec, cond= list(Hair=1))

## Not run:
## This will fail
ar_dist(hec, cond= list(Hair=c("Black", "Brown")))
ar_dist(hec, cond= list(Hair=1:2))

## End(Not run)
## But this will do the trick
a <- ar_slice(hec, slice=list(Hair=c("Black", "Brown")))
ar_dist(a, cond=~Hair)

## Combined
ar_dist(hec, marg=~Hair+Eye, cond=~Sex)
ar_dist(hec, marg=~Hair+Eye, cond="Sex")

ar_dist(hec, marg=~Hair+Eye, cond=list(Sex="Male"))
ar_dist(hec, marg=~Hair+Eye, cond=list(Sex="Male"), normalize=FALSE)

ar_dist(hec, cond=list(Sex="Male"))
ar_dist(hec, cond=list(Sex="Male"), normalize=FALSE)
```

`array-normalize` *Normalize an array*

Description

Normalize an array in various ways.

Usage

```
tabNormalize(tab, type = "none")

ar_normalize(tab, type = "none")
```

Arguments

tab	A multidimensional array
type	Either "none", "first", or "all"

Value

An array

Examples

```
ar_normalize( HairEyeColor, type="first")
ar_normalize( HairEyeColor, type="all")
```

---

array-operations *Operations on multidimensional arrays.*

Description

Operations like marginalize, permute, slicing etc on arrays A multidimensional table (an array) is here a vector with a dim and a dimnames attribute.

Usage

```
ar_perm(tab, perm)

ar_prod_list(lst)

ar_sum_list(lst)

ar_marg(tab, marg)

ar_equal(tab1, tab2, eps = 1e-12)

tab1 %a==% tab2

tab1 %a_% marg

tab1 %a^% extra

ar_expand(tab1, tab2)

ar_align(tab1, tab2)
```

Arguments

tab, tab1, tab2	Multidimensional arrays.
perm	A vector of indices or dimnames giving the desired permutiation.
lst	List of arrays.
marg	Specification of marginal; either a character vector, a numeric vector or a right hand sided formula For ar_perm and ar_marg it can also be a right hand sided formula.
eps	Criterion for checking equality of two arrays.
extra	List defining the extra dimensions.

Details

perm in ar_perm() can be a vector of indices (as in Rs own aperm()) but also a vector of dimnames. Currently there is no checking that the dimnames are actually in the array, so please take care.

Value

Most functions here return a multidimensional array.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

aperm, ar_perm, ar_slice, ar_slice_entries

Examples

```
ar1 <- array(1:8, dim=c(2,2,2), dimnames=list("a"=1:2,"b"=1:2,"c"=1:2))
ar2 <- array(1:8, dim=c(2,2,2), dimnames=list("b"=1:2,"c"=1:2,"d"=1:2))

## ## armarg ##
## Marginalize down to the bc-array
ar_marg(ar1, 2:3)
ar_marg(ar1, c("b","c"))
ar_marg(ar1, ~b + c)

## This gives an error
## ar_marg(ar1, c(2,5))
## ar_marg(ar1, c("b","w"))
## ar_marg(ar1, ~b + w)

## ## ar_perm ##
ar_perm(ar1, 1:3)      ## No change - an abc-table
ar_perm(ar1, c(2,3,1)) ## A bca-table
ar_perm(ar1, ~b + c + a)
```

```
## This gives error
## ar_perm(ar1, c(2,1))
## ar_perm(ar1, c(2,1,5))
## ar_perm(ar1, c(2,1,NA))

## ## ar_mult etc ##
## Multiply two arrays
out <- ar_mult(ar1, ar2)
out <- ar_perm(out, ~a + b + c + d) ## Just for comparison below
ftable(out)
## Alternative approch
df1 <- as.data.frame.table(ar1)
df2 <- as.data.frame.table(ar2)
df3 <- merge(df1, df2, by=c("b","c"))
df3 <- transform(df3, Freq=Freq.x*Freq.y)
ar3 <- xtabs(Freq ~ a + b + c + d, data=df3)
ftable(ar3)

## ## ar_expand ##
ar1.e <- ar_expand(ar1, ar2)
## ar1.e has dimnames b,c,d,a; values are simply replicated for each
## level of d.
dimnames(ar1.e)
ftable(ar1.e, row.vars="d")

## ## aralign ##
ar2.e <- ar_expand(ar2, ar1)
names(dimnames(ar2.e))
names(dimnames(ar1.e))
out <- ar_align(ar1.e, ar2.e)
names(dimnames(out)) ## Same as ar2.e


aa = ar_new(~a, levels=2, values=c(1,100))
ar_expand(aa, list(b=1:2))
aa %a^% list(b=1:2)

## ar_expand:
ar1 <- array(1:8, dim=c(2,2,2), dimnames=list("a"=1:2,"b"=1:2,"c"=1:2))
ar2 <- array(1:8, dim=c(2,2,2), dimnames=list("b"=1:2,"c"=1:2,"d"=1:2))

ar_expand(ar1, ar2) %>% ftable(row.vars=1) ## Same as
## ar_expand(ar1, dimnames(ar2)) %>% ftable(row.vars=1)
```

`array-properties` *Check if object is array*

Description

Check if object is array (that it is a vector with a dim attribute) and that the object has dimnames and that dimnames are named.

Usage

```
is.named.array(obj)

is_named_array_(obj)

is_number_vector_(obj)

is_dimnames_(obj)
```

Arguments

obj	Some R object.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
is.named.array( HairEyeColor )
is.named.array( matrix(1:4, nrow=2) )
is_named_array_( HairEyeColor )
is_named_array_( matrix(1:4, nrow=2) )
is_number_vector_(1:4)
is_number_vector_(list(1:4))
```

---

array-slice *Array slices*

Description

Functions for extracting slices of arrays

Usage

```
ar_slice(tab, slice = NULL, margin = names(slice), drop = TRUE,
  as.array = FALSE)

ar_slice_prim(tab, slice, drop = TRUE)

ar_slice_mult(tab, slice, val = 1, comp = 0)
```

```
ar_slice_entries(tab, slice, complement = FALSE)
```

Arguments

tab	An array with named dimnames.
slice	A list defining the slice.
margin	Names of variables in slice.
drop	If TRUE then dimensions with only one level will be dropped from the output.
as.array	If the resulting array is one-dimensional the result will by default be a vector with no dim attribute unless as.array is TRUE.
val	The values that entries in the slice will be multiplied with.
comp	The values that entries NOT in the slice will be multiplied with.
complement	If TRUE the complement of the entries are returned.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

ar_perm, ar_marg, ar_mult, ar_div, ar_add, ar_subt, ar_sum, ar_prod

Examples

```
x = HairEyeColor
s = list(Hair=c("Black","Brown"), Eye=c("Brown", "Blue"))

## arslice
s1 = ar_slice(x, slice=s)
s1

## ar_slice_entries
ar_slice_entries(x, slice=s)
ar_slice_entries(x, slice=s, complement=TRUE)

## ar_slice_mult
s2 = ar_slice_mult(x, slice=s)
s2

## arslice_prim does the same as arslice - faster, but the function is less
# flexible
sp = list(c(1,2), c(1,2), TRUE)
ar_slice_prim(x, slice=sp)
ar_slice(x, slice=s)
if ( require(microbenchmark) ){
  microbenchmark(ar_slice_prim(x, slice=sp), ar_slice(x, slice=s))
}
```

`ashtrees` *Crown dieback in ash trees*

Description

This dataset comes from a study of symptoms of crown dieback, cankers and symptoms caused by other pathogens and pests in ash trees (Fraxinus excelsior). In all 454 trees were observed in two plots. There are 8 categorical variables, 6 of which are binary and two are trichotomous with values representing increasing severity of symptoms, and one continuous variable, tree diameter at breast height (DBH).

Usage

```
ashtrees
```

Format

A data frame with 454 observations on the following 9 variables.

`plot` a factor with levels `2` `6`

`dieback` a factor with levels `0` `1` `2`

`dead50` a factor with levels `0` `0.5` `1`

`bushy` a factor with levels `0` `1`

`canker` a factor with levels `BRNCH` `MAIN` `NONE`

`wilt` a factor with levels `0` `1`

`roses` a factor with levels `0` `1`

`discolour` a factor with levels `0` `1`

`dbh` a numeric vector

References

Skovgaard JP, Thomsen IM, Skovgaard IM and Martinussen T (2009). Associations among symptoms of dieback in even-aged stands of ash (Fraxinus excelsior L.). Forest Pathology.

Examples

```
data(ashtrees)
head(ashtrees)
```

BodyFat *Body Fat Data*

Description

Estimates of the percentage of body fat determined by underwater weighing and various body circumference measurements for 252 men.

Usage

```
data(BodyFat)
```

Format

A data frame with 252 observations on the following 15 variables.

`Density` Density determined from underwater weighing, a numeric vector

`BodyFat` Percent body fat from Siri's (1956) equation, a numeric vector

`Age` in years, a numeric vector

`Weight` in lbs, a numeric vector

`Height` in inches, a numeric vector

`Neck` circumference in cm, a numeric vector

`Chest` circumference in cm, a numeric vector

`Abdomen` circumference in cm, a numeric vector

`Hip` circumference in cm, a numeric vector

`Thigh` circumference in cm, a numeric vector

`Knee` circumference in cm, a numeric vector

`Ankle` circumference in cm, a numeric vector

`Biceps` circumference in cm, a numeric vector

`Forearm` circumference in cm, a numeric vector

`Wrist` circumference in cm, a numeric vector

Source

For more information see http://lib.stat.cmu.edu/datasets/bodyfat

References

Bailey, Covert (1994). _Smart Exercise: Burning Fat, Getting Fit_, Houghton-Mifflin Co., Boston, pp. 179-186.

Behnke, A.R. and Wilmore, J.H. (1974). _Evaluation and Regulation of Body Build and Composition_, Prentice-Hall, Englewood Cliffs, N.J.

Siri, W.E. (1956), "Gross composition of the body", in _Advances in Biological and Medical Physics_, vol. IV, edited by J.H. Lawrence and C.A. Tobias, Academic Press, Inc., New York.

Katch, Frank and McArdle, William (1977). _Nutrition, Weight Control, and Exercise_, Houghton Mifflin Co., Boston.

Wilmore, Jack (1976). _Athletic Training and Physical Fitness: Physiological Principles of the Conditioning Process_, Allyn and Bacon, Inc., Boston.

Examples

```
data(BodyFat)
head(BodyFat)
```

---

breastcancer *Gene expression signatures for p53 mutation status in 250 breast cancer samples*

Description

Perturbations of the p53 pathway are associated with more aggressive and therapeutically refractory tumours. We preprocessed the data using Robust Multichip Analysis (RMA). Dataset has been truncated to the 1000 most informative genes (as selected by Wilcoxon test statistics) to simplify computation. The genes have been standardised to have zero mean and unit variance (i.e. z-scored).

Usage

```
breastcancer
```

Format

A data frame with 250 observations on 1001 variables. The first 1000 columns are numerical variables; the last column (named `code`) is a factor with levels `case` and `control`.

Details

The factor `code` defines whether there was a mutation in the p53 sequence (code=case) or not (code=control).

Source

Dr. Chris Holmes, c.holmes at stats dot. ox . ac .uk

References

Miller et al (2005, PubMed ID:16141321)

Examples

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

---

cad *Coronary artery disease data*

Description

A cross classified table with observational data from a Danish heart clinic. The response variable is CAD.

Usage

```
cad1
```

Format

A data frame with 236 observations on the following 14 variables.

`Sex` a factor with levels `Female` `Male`

`AngPec` a factor with levels `Atypical` `None` `Typical`

`AMI` a factor with levels `Definite` `NotCertain`

`QWave` a factor with levels `No` `Yes`

`QWavecode` a factor with levels `Nonusable` `Usable`

`STcode` a factor with levels `Nonusable` `Usable`

`STchange` a factor with levels `No` `Yes`

`SuffHeartF` a factor with levels `No` `Yes`

`Hypertrophi` a factor with levels `No` `Yes`

`Hyperchol` a factor with levels `No` `Yes`

`Smoker` a factor with levels `No` `Yes`

`Inherit` a factor with levels `No` `Yes`

`Heartfail` a factor with levels `No` `Yes`

`CAD` a factor with levels `No` `Yes`

Details

cad1: Complete dataset, 236 cases. cad2: Incomplete dataset, 67 cases. Information on (some of) the variables Hyperchol, Smoker, Inherit is missing.

cad1: Complete dataset, 236 cases. cad2: Incomplete dataset, 67 cases. Information on (some of) the variables Hyperchol, Smoker, Inherit is missing.

References

Højsgaard, Søren and Thiesson, Bo (1995). BIFROST - Block recursive models Induced From Relevant knowledge, Observations and Statistical Techniques. Computational Statistics and Data Analysis, vol. 19, p. 155-175

Hansen, J. F. (1980). The clinical diagnoisis of ichaeme heart disease du to coronary artery disease. Danish Medical Bulletin

Examples

```
data(cad1)
## maybe str(cad1) ; plot(cad1) ...
```

---

carcass *Lean meat contents of 344 pig carcasses*

Description

Measurement of lean meat percentage of 344 pig carcasses together with auxillary information collected at three Danish slaughter houses

Usage

```
carcass
```

Format

carcassall: A data frame with 344 observations on the following 17 variables.

`weight` Weight of carcass

`lengthc` Length of carcass from back toe to head (when the carcass hangs in the back legs)

`lengthf` Length of carcass from back toe to front leg (that is, to the shoulder)

`lengthp` Length of carcass from back toe to the pelvic bone

`Fat02, Fat03, Fat11, Fat12, Fat13, Fat14, Fat16` Thickness of fat layer at different locations on the back of the carcass (FatXX refers to thickness at (or rather next to) rib no. XX. Notice that 02 is closest to the head

`Meat11, Meat12, Meat13` Thickness of meat layer at different locations on the back of the carcass, see description above

`LeanMeat` Lean meat percentage determined by dissection

`slhouse` Slaughter house; a factor with levels `a` `b` `c`

`sex` Sex of the pig; a factor with `a` `b` `c`. Notice that it is no an error to have three levels; the third level refers to castrates

Note

carcass: Contains only the variables Fat11, Fat12, Fat13, Meat11, Meat12, Meat13, LeanMeat

Source

Busk, H., Olsen, E. V., Brøndum, J. (1999) Determination of lean meat in pig carcasses with the Autofom classification system, Meat Science, 52, 307-314

Examples

```
data(carcass)
head(carcass)
```

---

chestSim *Simulated data from the Chest Clinic example*

Description

Simulated data from the Chest Clinic example (also known as the Asia example) from Lauritzen and Spiegelhalter, 1988.

Usage

```
chestSim500
```

Format

A data frame with 500 observations on the following 8 variables.

`asia` a factor with levels `yes` `no`

`tub` a factor with levels `yes` `no`

`smoke` a factor with levels `yes` `no`

`lung` a factor with levels `yes` `no`

`bronc` a factor with levels `yes` `no`

`either` a factor with levels `yes` `no`

`xray` a factor with levels `yes` `no`

`dysp` a factor with levels `yes` `no`

References

Lauritzen and Spiegelhalter (1988) Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems (with Discussion). J. Roy. Stat. Soc. 50, p. 157-224.

Examples

```
data(chestSim500)
## maybe str(chestSim500) ; plot(chestSim500) ...
```

`combnPrim` *Generate All Combinations of n Elements Taken m at a Time*

Description

Generate all combinations of the elements of x taken m at a time. If x is a positive integer, returns all combinations of the elements of seq(x) taken m at a time.

Usage

```
combnPrim(x, m, simplify = TRUE)
```

Arguments

`x`	vector source for combinations, or integer n for x <- seq(n).
`m`	number of elements to choose.
`simplify`	logical indicating if the result should be simplified to a matrix; if FALSE, the function returns a list.

Value

A matrix or a list.

Note

The combnPrim function is a simplified version of the combn function. However, combnPrim is implemented in C and is considerably faster than combn.

Author(s)

P. T. Wallace and Søren Højsgaard

See Also

`combn`

Examples

```
x <- letters[1:20]
m <- 3

combn(x,m)
combnPrim(x,m)

combn(m,m)
combnPrim(m,m)

combn(x,m, simplify=FALSE)
combnPrim(x,m, simplify=FALSE)

system.time({ for (i in 1:100) { combnPrim(x,m) }})
system.time({ for (i in 1:100) { combn(x,m) }})

system.time({ for (i in 1:100) { combnPrim(x,m, simplify=FALSE) }})
system.time({ for (i in 1:100) { combn(x,m, simplify=FALSE) }})
```

`compareModels` *Generic function for model comparison*

Description

`compareModels` is a generic functions which invoke particular methods which depend on the class of the first argument

Usage

```
compareModels(object, object2, ...)
```

Arguments

`object, object2`	Model objects
`...`	Additional arguments

Value

The value returned depends on the class of the first argument.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

cov2pcor *Partial correlation (matrix)*

Description

cov2pcor calculates the partial correlation matrix from an (empirical) covariance matrix while conc2pcor calculates the partial correlation matrix from a concentration matrix (inverse covariance matrix).

Usage

```
cov2pcor(V)

conc2pcor(K)
```

Arguments

V	Covariance matrix
K	Concentration matrix

Value

A matrix with the same dimension as V.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
data(math)
S <- cov.wt(math)$cov
cov2pcor(S)
```

dietox *Growth curves of pigs in a 3x3 factorial experiment*

Description

The dietox data frame has 861 rows and 7 columns.

Usage

```
dietox
```

Format

This data frame contains the following columns: Weight, Feed, Time, Pig, Evit, Cu, Litter.

Source

Lauridsen, C., Højsgaard, S., Sørensen, M.T. C. (1999) Influence of Dietary Rapeseed Oli, Vitamin E, and Copper on Performance and Antioxidant and Oxidative Status of Pigs. J. Anim. Sci.77:906-916

Examples

```
data(dietox)
```

---

dumping *Gastric Dumping*

Description

A contingency table relating surgical operation, centre and severity of gastric dumping, a syndrome associated with gastric surgery.

Usage

```
dumping
```

Format

A 3x4x4 table of counts cross-classified by Symptom (none/slight/moderate), Operation (Vd/Va/Vh/Gr) and Centre (1:4).

Details

Gastric dumping syndrome is a condition where ingested foods bypass the stomach too rapidly and enter the small intestine largely undigested. It is an undesirable side-effect of gastric surgery. The table summarizes the results of a study comparing four different surgical operations on patients with duodenal ulcer, carried out in four centres, as described in Grizzle et al (1969). The four operations were: vagotomy and drainage, vagotomy and antrectomy (removal of 25% of gastric tissue), vagotomy and hemigastrectomy (removal of 50% of gastric tissue), and gastric restriction (removal of 75% of gastric tissue).

Source

Grizzle JE, Starmer CF, Koch GG (1969) Analysis of categorical data by linear models. Biometrics 25(3):489-504.

Examples

```
data(dumping)
plot(dumping)
```

---

graph-cliques *Get cliques of an undirected graph*

Description

Return a list of (maximal) cliques of an undirected graph.

Usage

```
getCliques(object)

## S3 method for class 'graphNEL'
getCliques(object)

## Default S3 method:
getCliques(object)

maxCliqueMAT(amat)
```

Arguments

object	An undirected graph represented either as a graphNEL object, a (dense) matrix, a (sparse) dgCMatrix
amat	An adjacency matrix.

Details

In graph theory, a clique is often a complete subset of a graph. A maximal clique is a clique which can not be enlarged. In statistics (and that is the convention we follow here) a clique is usually understood to be a maximal clique.

Finding the cliques of a general graph is an NP complete problem. Finding the cliques of triangualted graph is linear in the number of cliques.

The workhorse is the maxCliqueMAT function which calls the maxClique function in the RBGL package.

Value

A list.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

ug, dag, mcs, mcsMAT, rip, ripMAT, moralize, moralizeMAT

Examples

```
## graphNEL
uG1 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a)
getCliques(uG1)

## adjacency matrix
uG2 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a, result="matrix")
getCliques(uG2)

## adjacency matrix (sparse)
uG3 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a, result="Matrix")
getCliques(uG3)
```

---

graph-coerce *Graph coercion*

Description

Methods for changing graph representations

Usage

```
coerceGraph(object, result)

## S3 method for class 'graphNEL'
coerceGraph(object, result)

## S3 method for class 'matrix'
coerceGraph(object, result)

## S3 method for class 'dgCMatrix'
coerceGraph(object, result)

## S3 method for class 'igraph'
coerceGraph(object, result)
```

Arguments

object	A graph object
result	The desired output type

---

graph-create *Create undirected and directed graphs*

Description

These functions are wrappers for creation of graphs as implemented by graphNEL objects in the graph package.

Usage

```
ug(..., result = "graphNEL")

dag(..., result = "graphNEL", forceCheck = FALSE)
```

Arguments

result	The format of the graph. The possible choices are "graphNEL" (for a graphNEL object), "matrix" (for an adjacency matrix), "dgCMatrix" (for a sparse matrix), "igraph" (for an igraph object).
forceCheck	Logical determining if it should be checked if the graph is acyclical. Yes, one can specify graphs with cycles using the dag() function.
...	A generating class for a graph, see examples below

Value

Functions ug(), and dag() can return a graphNEL object, a sparse or dense adjacency matrix or an igraph object.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
## The following specifications of undirected graphs are equivalent:
uG1 <- ug(~ a:b:c + c:d)
uG2 <- ug(c("a","b","c"), c("c","d"))
uG3 <- ug(c("a","b"), c("a","c"), c("b","c"), c("c","d"))

graph::edges(uG1)
graph::nodes(uG1)
```

```
## The following specifications of directed acyclig graphs are equivalent:
daG1 <- dag(~ a:b:c + b:c + c:d)
daG2 <- dag(c("a","b","c"), c("b","c"), c("c","d"))

graph::edges(daG1)
graph::nodes(daG2)

## dag() allows to specify directed graphs with cycles:
daG4 <- dag(~ a:b + b:c + c:a) # A directed graph but with cycles
## A check for acyclicity can be done with
## daG5 <- dag(~ a:b + b:c + c:a, forceCheck=TRUE)

## A check for acyclicity is provided by topoSort
topoSort( daG2 )
topoSort( daG4 )

## Different representations
uG6 <- ug(~a:b:c + c:d, result="graphNEL")  # default
uG6
uG7 <- ug(~a:b:c + c:d, result="NEL")       # same
uG7
uG8 <- ug(~a:b:c + c:d, result="matrix")    # dense matrix
uG8
uG9 <- ug(~a:b:c + c:d, result="dgCMatrix") # sparse matrix
uG9
```

---

graph-edgeList *Find edges in a graph and edges not in a graph.*

Description

Returns the edges of a graph (or edges not in a graph) where the graph can be either a graphNEL object or an adjacency matrix.

Usage

```
edgeList(object, matrix = FALSE)

## Default S3 method:
edgeList(object, matrix = FALSE)

edgeListMAT(adjmat, matrix = FALSE)

nonEdgeList(object, matrix = FALSE)

## Default S3 method:
nonEdgeList(object, matrix = FALSE)

nonEdgeListMAT(adjmat, matrix = FALSE)
```

Arguments

object	A graphNEL object or an adjacency matrix.
matrix	If TRUE the result is a matrix; otherwise the result is a list.
adjmat	An adjacency matrix.

Examples

```
## A graph with edges
g  <- ug(~a:b + b:c + c:d)
gm <- graphNEL2M(g)
edgeList(g)
edgeList(gm)
edgeListMAT(gm)
edgeList(g, matrix=TRUE)
edgeList(gm, matrix=TRUE)
edgeListMAT(gm, matrix=TRUE)
nonEdgeList(g)
nonEdgeList(gm)
nonEdgeListMAT(gm)
## A graph without edges
g  <- ug(~a + b + c)
gm <- graphNEL2M(g)
edgeList(g)
edgeList(gm)
edgeListMAT(gm)
edgeList(g, matrix=TRUE)
edgeList(gm, matrix=TRUE)
edgeListMAT(gm, matrix=TRUE)
nonEdgeList(g)
nonEdgeList(gm)
nonEdgeListMAT(gm)
```

---

graph-gcproperties *Properties of a generating class (for defining a graph)*

Description

A set of generators define an undirected graph, here called a dependence graph. Given a set of generators it is checked 1) if the dependence dependence graph is in 1-1-correspondance with the genrators (such that the corresponding model is graphical) and 2) if the dependence graph is chordal (triangulated) (such that the corresponding model is decomposable).

Usage

```
isGraphical(x)

## Default S3 method:
isGraphical(x)

isDecomposable(x)

## Default S3 method:
isDecomposable(x)
```

Arguments

x	A generating class given as right hand sided formula or a list; see 'examples' below

Details

A set of sets of variables, say A_1, A_2, ... A_K is called a generating class for a graph with vertices V and edges E. If two variables a,b are in the same generator, say A_j, then a and b are vertices in the graph and there is an undirected edge between a and b.

The graph induced by `g1 = ~a:b + a:c + b:c + c:d` has edges `ab,ac,bc,cd`. The cliques of this graph are `abc,cd`. Hence there is not a 1-1-correspondance between the graph and the generators.

On the other hand, `g2 <- ~a:b:c + c:d` induces the same graph in this case there is a 1-1-correspondance.

The graph induced by `g3 <- ~a:b + b:c + c:d + d:a` is in 1-1-correspondance with its dependence graph, but the graph is not chordal.

Value

TRUE or FALSE

Author(s)

Søren Højsgaard, `<sorenh@math.aau.dk>`

See Also

`mcs`, `rip`

Examples

```
g1 <- ~a:b+a:c+b:c+c:d
g2 <- ~a:b:c+c:d
g3 <- ~a:b + b:c + c:d + d:a

isGraphical( g1 ) # FALSE
isGraphical( g2 ) # TRUE
```

```
isGraphical( g3 ) # TRUE

isDecomposable( g1 ) # FALSE
isDecomposable( g2 ) # TRUE
isDecomposable( g3 ) # TRUE

## A generating class can be given as a list:
f <- list(c("a","b"), c("b","c"), c("a","c"))
isGraphical( f )
isDecomposable( f )
```

---

graph-iplot *Function for plotting graphs using the 'igraph' package.*

Description

Generic function for plotting graphs using the 'igraph' package and a plot method for graphNEL objects.

Usage

```
iplot(x, ...)

## S3 method for class 'graphNEL'
iplot(x, ...)
```

Arguments

x	A graph object to be plotted.
...	Additional arguments

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
UG <- ug(~a:b+b:c:d)
iplot(UG)
```

graph-is *Check properties of graphs.*

Description

Check if a graph is 1) a directed acyclic graph (DAG), 2) a directed graph (DG), 3) an undirected graph (UG), 4) a triangulated (chordal) undirected graph (TUG). This is done for graphs represented as 1) graphNEL (from the graph package), 2) an adjacency matrix, 3) a sparse adjacency matrix (a dgCMatrix from the Matrix package).

Usage

```
is.DAG(object)

## S3 method for class 'graphNEL'
is.DAG(object)

## Default S3 method:
is.DAG(object)

is.DAGMAT(object)

is.UG(object)

## S3 method for class 'graphNEL'
is.UG(object)

## Default S3 method:
is.UG(object)

is.UGMAT(object)

is.TUG(object)

## S3 method for class 'graphNEL'
is.TUG(object)

## Default S3 method:
is.TUG(object)

is.TUGMAT(object)

is.DG(object)

## S3 method for class 'graphNEL'
is.DG(object)
```

```
## Default S3 method:
is.DG(object)

is.DGMAT(object)
```

Arguments

object	A graph represented as 1) graphNEL (from the graph package), 2) an adjacency matrix, 3) a sparse adjacency matrix (a dgCMatrix from the Matrix package).

Details

A non-zero value at entry (i,j) in an adjacency matrix A for a graph means that there is an edge from i to j. If also (j,i) is non-zero there is also an edge from j to i. In this case we may think of a bidirected edge between i and j or we may think of the edge as being undirected. We do not distinguish between undirected and bidirected edges in the gRbase package. On the other hand, graphNEL objects from the graph package makes such a distinction (the function `edgemode()` will tell if edges are "directed" or "undirected" in a graphNEL object).

The function `is.UG()` checks if the adjacency matrix is symmetric (If applied to a graphNEL, the adjacency matrix is created and checked for symmetry.)

The function `is.TUG()` checks if the graph is undirected and triangulated (also called chordal) by checking if the adjacency matrix is symmetric and the vertices can be given a perfect ordering using maximum cardinality seach.

The function `is.DG()` checks if a graph is directed, i.e., that there are no undirected edges. This is done by computing the elementwise product of A and the transpose of A; if there are no non–zero entries in this product then the graph is directed.

The function `is.DAG()` will return `TRUE` if all edges are directed and if there are no cycles in the graph. (This is checked by checking if the vertices in the graph can be given a topological ordering which is based on identifying an undirected edge with a bidrected edge).

There is a special case, namely if the graph has no edges at all (such that the adjacency matrix consists only of zeros). Such a graph is both undirected, triangulated, directed and directed acyclic.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

dag, ug

Examples

```
## DAGs
dagNEL  <- dag(~ a:b:c + c:d:e, result="NEL")
dagMAT  <- dag(~ a:b:c + c:d:e, result="matrix")
dagMATS <- dag(~ a:b:c + c:d:e, result="Matrix")
```

```
## Undirected graphs
ugNEL  <- ug(~a:b:c + c:d:e, result="NEL")
ugMAT  <- ug(~a:b:c + c:d:e, result="matrix")
ugMATS <- ug(~a:b:c + c:d:e, result="Matrix")

## Is it a DAG?
is.DAG(dagNEL)
is.DAG(dagMAT)
is.DAG(dagMATS)

is.DAG(ugNEL)
is.DAG(ugMAT)
is.DAG(ugMATS)

## Is it an undirected graph
is.UG(dagNEL)
is.UG(dagMAT)
is.UG(dagMATS)

is.UG(ugNEL)
is.UG(ugMAT)
is.UG(ugMATS)

## Is it a triangulated (i.e. chordal)  undirected graph
is.TUG(dagNEL)
is.TUG(dagMAT)
is.TUG(dagMATS)

is.TUG(ugNEL)
is.TUG(ugMAT)
is.TUG(ugMATS)

## Example where the graph is not triangulated
ug2NEL  <- ug(~ a:b + b:c + c:d + d:a, result="NEL")
ug2MAT  <- ug(~ a:b + b:c + c:d + d:a, result="matrix")
ug2MATS <- ug(~ a:b + b:c + c:d + d:a, result="Matrix")

is.TUG(ug2NEL)
is.TUG(ug2MAT)
is.TUG(ug2MATS)

## Bidirected graphs
graph::edgemode(ugNEL)
graph::edgemode(ugNEL) <- "directed"
graph::edgemode(ugNEL)
is.DAG(ugNEL)
is.UG(ugNEL)
```

---

graph-mcs *Maximum cardinality search on undirected graph.*

Description

Returns (if it exists) a perfect ordering of the vertices in an undirected graph.

Usage

```
mcs(object, root = NULL, index = FALSE)

## Default S3 method:
mcs(object, root = NULL, index = FALSE)

mcsMAT(amat, vn = colnames(amat), root = NULL, index = FALSE)

mcsmarked(object, discrete = NULL, index = FALSE)

## Default S3 method:
mcsmarked(object, discrete = NULL, index = FALSE)

mcsmarkedMAT(amat, vn = colnames(amat), discrete = NULL, index = FALSE)
```

Arguments

object	An undirected graph represented either as a graphNEL object, an igraph, a (dense) matrix, a (sparse) dgCMatrix.
root	A vector of variables. The first variable in the perfect ordering will be the first variable on 'root'. The ordering of the variables given in 'root' will be followed as far as possible.
index	If TRUE, then a permutation is returned
amat	Adjacency matrix
vn	Nodes in the graph given by adjacency matrix
discrete	A vector indicating which of the nodes are discrete. See 'details' for more information.

Details

An undirected graph is decomposable iff there exists a perfect ordering of the vertices. The maximum cardinality search algorithm returns a perfect ordering of the vertices if it exists and hence this algorithm provides a check for decomposability. The mcs() functions finds such an ordering if it exists.

The notion of strong decomposability is used in connection with e.g. mixed interaction models where some vertices represent discrete variables and some represent continuous variables. Such graphs are said to be marked. The mcsmarked() function will return a perfect ordering iff the graph is strongly decomposable. As graphs do not know about whether vertices represent discrete or continuous variables, this information is supplied in the discrete argument.

Value

A vector with a linear ordering (obtained by maximum cardinality search) of the variables or character(0) if such an ordering can not be created.

Note

The workhorse is the mcsMAT function.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

moralize, jTree, rip, ug, dag

Examples

```
uG <- ug(~ me:ve + me:al + ve:al + al:an + al:st + an:st)
mcs(uG)
mcsMAT( graphNEL2M(uG) )
## Same as
uG <- ug(~ me:ve + me:al + ve:al + al:an + al:st + an:st, result="matrix")
mcsMAT(uG)

## Marked graphs
uG1 <- ug(~ a:b + b:c + c:d)
uG2 <- ug(~ a:b + a:d + c:d)
## Not strongly decomposable:
mcsmarked(uG1, discrete=c("a","d"))
## Strongly decomposable:
mcsmarked(uG2, discrete=c("a","d"))
```

---

graph-minimaltriang *Minimal triangulation of an undirected graph*

Description

An undirected graph uG is triangulated (or chordal) if it has no cycles of length >= 4 without a chord which is equivalent to that the vertices can be given a perfect ordering. Any undirected graph can be triangulated by adding edges to the graph, so called fill-ins which gives the graph TuG. A triangulation TuG is minimal if no fill-ins can be removed without breaking the property that TuG is triangulated.

Usage

```
minimalTriang(object, tobject = triangulate(object), result = NULL,
  details = 0)

## Default S3 method:
minimalTriang(object, tobject = triangulate(object),
```

```
  result = NULL, details = 0)

minimalTriangMAT(amat, tamat = triangulateMAT(amat), details = 0)
```

Arguments

object	An undirected graph represented either as a graphNEL object, a (dense) matrix, a (sparse) dgCMatrix.
tobject	Any triangulation of object; must be of the same representation.
result	The type (representation) of the result. Possible values are "graphNEL", "matrix", "dgCMatrix". Default is the same as the type of object.
details	The amount of details to be printed.
amat	The undirected graph which is to be triangulated; a symmetric adjacency matrix.
tamat	Any triangulation of object; a symmetric adjacency matrix.

Details

For a given triangulation tobject it may be so that some of the fill-ins are superflous in the sense that they can be removed from tobject without breaking the property that tobject is triangulated. The graph obtained by doing so is a minimal triangulation.

Notice: A related concept is the minimum triangulation, which is the the graph with the smallest number of fill-ins. The minimum triangulation is unique. Finding the minimum triangulation is NP-hard.

Value

minimalTriang() returns a graphNEL object while minimalTriangMAT() returns an adjacency matrix.

Author(s)

Clive Bowsher <C.Bowsher@statslab.cam.ac.uk> with modifications by Søren Højsgaard, <sorenh@math.aau.dk>

References

Kristian G. Olesen and Anders L. Madsen (2002): Maximal Prime Subgraph Decomposition of Bayesian Networks. IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, PART B: CYBERNETICS, VOL. 32, NO. 1, FEBRUARY 2002

See Also

mpd, rip, triangulate

Examples

```
## A graphNEL object
g1 <- ug(~a:b+b:c+c:d+d:e+e:f+a:f+b:e)
x <- minimalTriang(g1)

## g2 is a triangulation of g1 but it is not minimal
g2 <- ug(~a:b:e:f+b:c:d:e)
x<-minimalTriang(g1, tobject=g2)

## An adjacency matrix
g1m <- ug(~a:b+b:c+c:d+d:e+e:f+a:f+b:e, result="matrix")
x<-minimalTriangMAT(g1m)
```

---

graph-moralize *Moralize a directed acyclic graph*

Description

Moralize a directed acyclic graph which means marrying parents and dropping directions.

Usage

```
moralize(object, ...)

## Default S3 method:
moralize(object, result = NULL, ...)
```

Arguments

object	A directed acyclic graph represented either as a graphNEL object, an igraph, a (dense) matrix, a (sparse) dgCMatrix.
result	The representation of the moralized graph. When NULL the representation will be the same as the input object.
...	Additional arguments, currently not used

Value

A moralized graph represented either as a graphNEL, a dense matrix or a sparse dgCMatrix.

Note

The workhorse is the moralizeMAT function.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

mcs, jTree, rip, ug, dag

Examples

```
daG <- dag(~me+ve,~me+al,~ve+al,~al+an,~al+st,~an+st)
moralize(daG)

daG <- dag(~me+ve,~me+al,~ve+al,~al+an,~al+st,~an+st, result="matrix")
moralizeMAT(daG)

if (require(igraph)){
M <- matrix(c(1,2,3,3), nrow=2)
G <- graph.edgelist(M)
G
V(G)$name
moralize(G)
}
```

---

graph-mpd *Maximal prime subgraph decomposition*

Description

Finding a junction tree representation of the MPD (maximal prime subgraph decomposition) of an undirected graph The maximal prime subgraph decomposition of a graph is the smallest subgraphs into which the graph can be decomposed.

Usage

```
mpd(object, tobject = minimalTriang(object), details = 0)

## Default S3 method:
mpd(object, tobject = triangulate(object), details = 0)

mpdMAT(amat, tamat = minimalTriangMAT(amat), details = 0)
```

Arguments

object	An undirected graph; a graphNEL object
tobject	Any minimal triangulation of object; a graphNEL object
details	The amount of details to be printed.
amat	An undirected graph; a symmetric adjacency matrix
tamat	Any minimal triangulation of object; a symmetric adjacency matrix

Value

A list with components "nodes", "cliques", "separators", "parents", "children", "nLevels". The component "cliques" defines the subgraphs.

Author(s)

Clive Bowsher <C.Bowsher@statslab.cam.ac.uk> with modifications by Søren Højsgaard, <sorenh@math.aau.dk>

References

Kristian G. Olesen and Anders L. Madsen (2002): Maximal Prime Subgraph Decomposition of Bayesian Networks. IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, PART B: CYBERNETICS, VOL. 32, NO. 1, FEBRUARY 2002

See Also

mcs, mcsMAT, minimalTriang, minimalTriangMAT, rip, ripMAT, triangulate, triangulateMAT

Examples

```
## Maximal prime subgraph decomposition - a graphNEL object
g1 <- ug(~ a:b + b:c + c:d + d:e + e:f + a:f + b:e)
if (interactive()) plot(g1)
x <- mpd(g1)

## Maximal prime subgraph decomposition - an adjacency matrix
g1m <- ug(~ a:b + b:c + c:d + d:e + e:f + a:f + b:e, result="matrix")
if (interactive()) plot(as(g1m, "graphNEL"))
x <- mpdMAT(g1m)
```

---

graph-query *Query a graph*

Description

Unified approach to query a graph about its properties

Usage

```
querygraph(object, op, set = NULL, set2 = NULL, set3 = NULL)

ancestors(set, object)

ancestralSet(set, object)

parents(set, object)
```

```
children(set, object)

closure(set, object)

simplicialNodes(object)

ancestralGraph(set, object)

is.complete(object, set = NULL)

is.decomposition(set, set2, set3, object)

is.simplicial(set, object)
```

Arguments

object	A graph
op	The operation or query
set, set2, set3	Sets of nodes in graph

---

graph-randomdag *Random directed acyclic graph*

Description

Generate a random directed acyclic graph (DAG)

Usage

```
random_dag(V, maxpar = 3, wgt = 0.1)
```

Arguments

V	The set of vertices.
maxpar	The maximum number of parents each node can have
wgt	A parameter controlling how likely it is for a node to have a certain number of parents; see 'Details'

Details

If the maximum number of parents for a node is, say 3 and wgt=0.1, then the probability of the node ending up with 0,1,2,3 parents is proportional to 0.1^0, 0.1^1, 0.1^2, 0.1^3.

Value

A graphNEL object.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
dg   <- random_dag(1:1000, maxpar=5, wgt=.9)
table(sapply(vpar(dg),length))

dg   <- random_dag(1:1000, maxpar=5, wgt=.5)
table(sapply(vpar(dg),length))

dg   <- random_dag(1:1000, maxpar=5, wgt=.1)
table(sapply(vpar(dg),length))
```

---

graph-rip *Create RIP ordering of the cliques of an undirected graph; create junction tree.*

Description

A RIP (running intersection property) ordering of the cliques is also called a perfect ordering. If the graph is not chordal, then no such ordering exists.

Usage

```
rip(object, ...)

## Default S3 method:
rip(object, root = NULL, nLevels = NULL, ...)

ripMAT(amat, root = NULL, nLevels = rep(2, ncol(amat)))

jTree(object, ...)

## Default S3 method:
jTree(object, nLevels = NULL, ...)

jTreeMAT(amat, nLevels = rep(2, ncol(amat)), ...)
```

Arguments

object	An undirected graph represented either as a graphNEL object, an igraph, a (dense) matrix, a (sparse) dgCMatrix.
...	Additional arguments; currently not used
root	A vector of variables. The first variable in the perfect ordering will be the first variable on 'root'. The ordering of the variables given in 'root' will be followed as far as possible.
nLevels	Typically, the number of levels of the variables (nodes) when these are discrete. Used in determining the triangulation using a "minimum clique weight heuristic". See section 'details'.
amat	Adjacency matrix

Details

The RIP ordering of the cliques of a decomposable (i.e. chordal) graph is obtained by first ordering the variables linearly with maximum cardinality search (by mcs). The root argument is transfered to mcs as a way of controlling which clique will be the first in the RIP ordering. The jTree() (and jTree()) (for "junction tree") is just a wrapper for a call of triangulate() followed by a call of rip().

Value

rip returns a list (an object of class ripOrder. A print method exists for such objects.)

Note

The workhorse is the ripMAT() function. The nLevels argument to the rip functions has no meaning.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

mcs triangulate moralize ug, dag

Examples

```
## graphNEL
uG <- ug(~me:ve + me:al + ve:al + al:an + al:st + an:st)
mcs(uG)
rip(uG)
jTree(uG)

## Adjacency matrix
uG <- ug(~me:ve:al + al:an:st, result="matrix")
mcs(uG)
```

```
rip(uG)
jTree(uG)

## Sparse adjacency matrix
uG <- ug(c("me", "ve", "al"), c("al", "an", "st"), result="Matrix")
mcs(uG)
rip(uG)
jTree(uG)

## Non--decomposable graph
uG <- ug(~1:2 + 2:3 + 3:4 + 4:5 + 5:1)
mcs(uG)
rip(uG)
jTree(uG)
```

---

graph-toposort *Topological sort of vertices in directed acyclic graph*

Description

A topological ordering of a directed graph is a linear ordering of its vertices such that, for every edge (u->v), u comes before v in the ordering. A topological ordering is possible if and only if the graph has no directed cycles, that is, if it is a directed acyclic graph (DAG). Any DAG has at least one topological ordering. Can hence be used for checking if a graph is a DAG.

Usage

```
topoSort(object, index = FALSE)

## Default S3 method:
topoSort(object, index = FALSE)

topoSortMAT(amat, index = FALSE)
```

Arguments

object	An graph represented either as a graphNEL object, an igraph, a (dense) matrix, a (sparse) dgCMatrix.
index	If FALSE, an ordering is returned if it exists and character(0) otherwise. If TRUE, the index of the variables in an adjacency matrix is returned and -1 otherwise.
amat	Adjacency matrix.

Value

If FALSE, an ordering is returned if it exists and character(0) otherwise. If TRUE, the index of the variables in an adjacency matrix is returned and -1 otherwise.

Note

The workhorse is the `topoSortMAT` function which takes an adjacency matrix as input

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

dag, ug

Examples

```
dagMAT  <- dag(~a:b:c+c:d:e, result="matrix")
dagMATS <- as(dagMAT, "dgCMatrix")
dagNEL  <- as(dagMAT, "graphNEL")

topoSort(dagMAT)
topoSort(dagMATS)
topoSort(dagNEL)
```

---

graph-triang *Triangulation of an undirected graph*

Description

This function will triangulate an undirected graph by adding fill-ins.

Usage

```
triang(object, ...)

## Default S3 method:
triang(object, control = list(), ...)

triang_mcwh(object, ...)

## Default S3 method:
triang_mcwh(object, nLevels = NULL, result = NULL,
  check = TRUE, ...)

triang_elo(object, ...)

## Default S3 method:
triang_elo(object, order = NULL, result = NULL,
  check = TRUE, ...)
```

```
triang_eloMAT(amat, order = NULL)

triang_eloMAT_(amat, order)
```

Arguments

object	An undirected graph represented either as a graphNEL object, an igraph, a (dense) matrix, a (sparse) dgCMatrix.
...	Additional arguments, currently not used.
control	A list controlling the triangulation; see 'examples'.
nLevels	The number of levels of the variables (nodes) when these are discrete. Used in determining the triangulation using a "minimum clique weight heuristic". See section 'details'.
result	The type (representation) of the result. Possible values are "graphNEL", "igraph", "matrix", "dgCMatrix". Default is the same as the type of object.
check	If TRUE (the default) it is checked whether the graph is triangulated before doing the triangulation; gives a speed up if FALSE
order	Elimation order; a character vector or numeric vector.
amat	Adjacency matrix; a (dense) matrix, or a (sparse) dgCMatrix.

Details

The triangulation is made so as the total state space is kept low by applying a minimum clique weight heuristic: When a fill-in is necessary, the algorithm will search for an edge to add such that the complete set to be formed will have as small a state-space as possible. It is in this connection that the nLevels values are used.

Default (when nLevels=NULL) is to take nLevels=2 for all nodes. If nLevels is the same for all nodes then the heuristic aims at keeping the clique sizes small.

Value

A triangulated graph represented either as a graphNEL, a (dense) matrix or a (sparse) dgCMatrix.

Note

Care should be taken when specifying nLevels for other representations than adjacency matrices: Since the triangulateMAT function is the workhorse, any other representation is transformed to an adjacency matrix and the order of values in nLevels most come in the order of the nodes in the adjacency matrix representation.

Currently there is no check for that the graph is undirected.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

ug dag mcs, mcsMAT rip, ripMAT, moralize, moralizeMAT

Examples

```
## graphNEL
uG1 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a)
uG2 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a, result="matrix")
uG3 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a, result="Matrix")

## Default triangulation: minimum clique weight heuristic
# (default is that each node is given the same weight):

tuG1 <- triang(uG1)
## Same as
triang_mcwh(uG1)

## Alternative: Triangulation from a desired elimination order
# (default is that the order is order of the nodes in the graph):

triang(uG1, control=list(method="elo"))
## Same as:
triang_elo(uG1)

## More control: Define the number of levels for each node:
tuG1 <- triang(uG1, control=list(method="mcwh", nLevels=c(2, 3, 2, 6, 4, 9)))
tuG1 <- triang_mcwh(uG1, nLevels=c(2, 3, 2, 6, 4, 9))

tuG1 <- triang(uG1, control=list(method="elo", order=c("a", "e", "f")))
tuG1 <- triang_elo(uG1, order=c("a", "e", "f"))
```

---

graph-triangulate *Triangulation of an undirected graph*

Description

This function will triangulate an undirected graph by adding fill-ins.

Usage

```
triangulate(object, ...)

## Default S3 method:
triangulate(object, nLevels = NULL, result = NULL,
  check = TRUE, ...)

triangulateMAT(amat, nLevels = rep(2, ncol(amat)), ...)
```

Arguments

object	An undirected graph represented either as a graphNEL object, an igraph, a (dense) matrix, a (sparse) dgCMatrix.
...	Additional arguments, currently not used.
nLevels	The number of levels of the variables (nodes) when these are discrete. Used in determining the triangulation using a "minimum clique weight heuristic". See section 'details'.
result	The type (representation) of the result. Possible values are "graphNEL", "igraph", "matrix", "dgCMatrix". Default is the same as the type of object.
check	If TRUE (the default) it is checked whether the graph is triangulated before doing the triangulation; gives a speed up if FALSE
amat	Adjacency matrix; a (dense) matrix, or a (sparse) dgCMatrix.

Details

The workhorse is the triangulateMAT function.

The triangulation is made so as the total state space is kept low by applying a minimum clique weight heuristic: When a fill-in is necessary, the algorithm will search for an edge to add such that the complete set to be formed will have as small a state-space as possible. It is in this connection that the nLevels values are used.

Default (when nLevels=NULL) is to take nLevels=2 for all nodes. If nLevels is the same for all nodes then the heuristic aims at keeping the clique sizes small.

Value

A triangulated graph represented either as a graphNEL, a (dense) matrix or a (sparse) dgCMatrix.

Note

Care should be taken when specifying nLevels for other representations than adjacency matrices: Since the triangulateMAT function is the workhorse, any other representation is transformed to an adjacency matrix and the order of values in nLevels most come in the order of the nodes in the adjacency matrix representation.

Currently there is no check for that the graph is undirected.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

ug, dag, mcs, mcsMAT, rip, ripMAT, moralize, moralizeMAT

Examples

```
## graphNEL
uG1 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a)
tuG1 <- triangulate(uG1)

## adjacency matrix
uG2 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a, result="matrix")
tuG2 <- triangulate(uG2)

## adjacency matrix (sparse)
uG2 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a, result="Matrix")
tuG2 <- triangulate(uG2)
```

---

graph-vpar *List of vertices and their parents for graph.*

Description

Get list of vertices and their parents for graph.

Usage

```
vchi(object, getv = TRUE, forceCheck = TRUE)

vchiMAT(object, getv = TRUE, forceCheck = TRUE)

## S3 method for class 'graphNEL'
vchi(object, getv = TRUE, forceCheck = TRUE)

## S3 method for class 'Matrix'
vchi(object, getv = TRUE, forceCheck = TRUE)

## S3 method for class 'matrix'
vchi(object, getv = TRUE, forceCheck = TRUE)

vpar(object, getv = TRUE, forceCheck = TRUE)

vparMAT(object, getv = TRUE, forceCheck = TRUE)

## S3 method for class 'graphNEL'
vpar(object, getv = TRUE, forceCheck = TRUE)

## S3 method for class 'Matrix'
vpar(object, getv = TRUE, forceCheck = TRUE)

## S3 method for class 'matrix'
vpar(object, getv = TRUE, forceCheck = TRUE)
```

Arguments

object	An object representing a graph. Valid objects are an adjacency matrix or as a graphNEL.
getv	The result is by default a list of vectors of the form (v, pa1, pa2, ... paN) where pa1, pa2, ... paN are the parents of v. If getv is FALSE then the vectors will have the form (pa1, pa2, ... paN)
forceCheck	Logical indicating if it should be checked that the object is a DAG.

Value

A list of vectors where each vector will have the form (v, pa1, pa2, ... paN) where pa1, pa2, ... paN are the parents of v.

See Also

dag, ug

Examples

```
## DAGs
dagMAT <- dag(~a:b:c + c:d:e, result="matrix")
dagNEL <- dag(~a:b:c + c:d:e, result="NEL")
vpar(dagMAT)
vpar(dagNEL)
vpar(dagMAT, getv=FALSE)
vpar(dagNEL, getv=FALSE)
## Undirected graphs
ugMAT <- ug(~a:b:c + c:d:e, result="matrix")
ugNEL <- ug(~a:b:c + c:d:e, result="NEL")
## Not run:
## This will fail because the adjacency matrix is symmetric and the
## graphNEL has undirected edges
vpar(ugMAT)
vpar(ugNEL)

## End(Not run)
## When forceCheck is FALSE, it will not be detected that the graphs are undirected.
vpar(ugMAT, forceCheck=FALSE)
vpar(ugNEL, forceCheck=FALSE)
## Bidirected graphs
## This is, for graphNELs, the same as working with bidirected edges:
if (require(graph)){
graph::edgemode(ugNEL)
graph::edgemode(ugNEL) <- "directed"
graph::edgemode(ugNEL)
vpar(ugNEL,FALSE)
}
```

graph-xxx2yyy *Graph, matrix and generating class coercions*

Description

Graph and matrix coercions where speed is an issue.

Usage

```
graphNEL2M(gn, result = "matrix")

graphNEL2MAT(gn, limit = 100)

graphNEL2ftM(gn)

graphNEL2tfM(gn)

graphNEL2igraph(gn)

M2igraph(amat)

M2graphNEL(amat)

M2adjList(amat)

M2ugList(amat)

M2dagList(amat)

ugList2graphNEL(glist, vn = NULL)

ugList2M(glist, vn = NULL, result = "matrix")

dagList2graphNEL(glist, vn = NULL)

dagList2M(glist, vn = NULL, result = "matrix")

adjList2M(alist, result = "matrix")

dagList2tfM(glist)

as.adjMAT(gn, result = "matrix")

ug2dag(gn)

MAT2matrix(mat)
```

```
MAT2dgCMatrix(mat)
```

Arguments

gn	A graphNEL object
result	Either "matrix" or "dgCMatrix" (for a sparse matrix representation)
limit	If number of nodes is larger than limit, the result will be a sparse dgCMatrix; otherwise a dense matrix.
amat	Adjacency matrix
glist	A list of generators where a generator is a character vector. If interpreted as generators of an undirected graph, a generator is a complete set of vertices in the graph. If interpreted as generators of a dag, a generator (v1,...,vn) means that there will be arrows from v2,...,vn to v1.
vn	The names of the vertices in the graphs. These will be the row and column names of the matrix
alist	An adjacency list.
mat	Either a dense matrix or a sparse dgCMatrix.

Value

An adjacency matrix (or NULL if glist has length 0)

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

ug, dag

Examples

```
glist <- list(1:3, 2:4, 4:5)
am1 <- ugList2M( glist )
am2 <- dagList2M( glist )
if (interactive()){
  plot(as(am1, "graphNEL"))
  plot(as(am2, "graphNEL"))
}


## Sparse and dense adjacency matrices converted to igraph
g1 <- ug(~a:b + b:c + c:d, result="matrix")
g2 <- ug(~a:b + b:c + c:d, result="dgCMatrix")
plot( M2igraph( g1 ) )
plot( M2igraph( g2 ) )

## Sparse and dense adjacency matrices converted to graphNEL
```

```
g1 <- ug(~a:b + b:c + c:d, result="matrix")
g2 <- ug(~a:b + b:c + c:d, result="dgCMatrix")
M2graphNEL( g1 )
M2graphNEL( g2 )

## Sparse and dense adjacency matrices converted to adjacency list
g1 <- ug(~a:b + b:c + c:d, result="matrix")
g2 <- ug(~a:b + b:c + c:d, result="dgCMatrix")
M2adjList( g1 )
## M2adjList( g2 ) FAILS for sparse matrix

## Sparse and dense adjacency matrices converted to cliques
g1 <- ug(~a:b + b:c + c:d, result="matrix")
g2 <- ug(~a:b + b:c + c:d, result="dgCMatrix")
M2ugList( g1 )
M2ugList( g2 )

## Sparse and dense adjacency matrices converted to cliques
g1 <- ug(~a:b + b:c + c:d, result="matrix")
g2 <- ug(~a:b + b:c + c:d, result="dgCMatrix")
M2dagList( g1 )
## M2dagList( g2 ) ## Fails for sparse matrix

g1 <- dag(~a:b + b:c + c:d, result="matrix")
g2 <- dag(~a:b + b:c + c:d, result="dgCMatrix")
M2dagList( g1 )
## M2dagList( g2 ) ## FIXME Fails for sparse matrix
```

---

gRbase *The package 'gRbase': summary information*

Description

This package provides a basis for graphical modelling in R and in particular for other graphical modelling packages, most notably **gRim**, **gRain** and **gRc**.

Details

gRbase provides the following:

- Implementation of various graph algorithms, including maximum cardinality search, maximal prime subgraph decomposition, triangulation. See the vignette `graphs`.
- Implementation of various "high level" array operations, including multiplication/division, marginalization, slicing, permutation. See the vignette `ArrayOps`.
- Implementation of various "low level" array operations. See the vignette `ArrayOpsPrim`.
- A collection of datasets
- A general framework for setting up data and model structures and provide examples for fitting hierarchical log linear models for contingency tables and graphical Gaussian models for the multivariate normal distribution. (Notice: This last part is not maintained / developed further.)

Authors

Soren Hojsgaard, Department of Mathematical Sciences, Aalborg University, Fredrik Bajersvej 7G, DK-9220 Aalborg East, Denmark

Contributions from Claus Dethlefsen, Clive Bowsher, David Edwards.

Acknowledgements

Thanks to the other members of the gR initiative, in particular to David Edwards for providing functions for formula-manipulation.

References

Hojsgaard, S., Edwards, D., Lauritzen, S. (2012) Graphical models with R. Springer. ISBN: 978-1-4614-2298-3

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

`gRbase-generics` *Compile and propagate functions*

Description

`compile` and `propagate` are generic functions which invoke particular methods which depend on the class of the first argument

Usage

```
fit(object, ...)

compile(object, ...)

propagate(object, ...)

stepwise(object, ...)
```

Arguments

`object`	An object
`...`	Additional arguments which depends on the class of the object

Value

The value returned depends on the class of the first argument.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Højsgaard, Søren; Edwards, David; Lauritzen, Steffen (2012): Graphical Models with R, Springer

See Also

compile.grain, propagate.grain

---

gRbase-utilities *Utility functions for gRbase*

Description

Utility functions for gRbase package. Includes 'faster versions' of certain standard R functions.

Details

colwiseProd multiplies a vector and a matrix columnwise (as opposed to rowwise which is achieved by v*M). Hence colwiseProd does the same as t(v*t(M)) - but it does so faster for numeric values.

Value

A vector or a logical.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
## colwiseProd
M <- matrix(1:16, nrow=4)
v <- 1:4

t(v*t(M))
colwiseProd(v,M)

system.time(for (ii in 1:100000)  t(v*t(M)))
system.time(for (ii in 1:100000)  colwiseProd(v,M))
```

lizard *Lizard behaviour*

Description

In a study of lizard behaviour, characteristics of 409 lizards were recorded, namely species (S), perch diameter (D) and perch height (H). The focus of interest is in how the propensities of the lizards to choose perch height and diameter are related, and whether and how these depend on species.

Usage

```
lizard
```

Format

A 3–dimensional array with factors diam: "<=4" ">4" height: ">4.75" "<=4.75" species: "anoli" "dist"

References

Schoener TW (1968) The anolis lizards of bimini: Resource partitioning in a complex fauna. Ecology 49:704-726

Examples

```
data(lizard)

# Datasets lizardRAW and lizardDF are generated with the following code
#lizardAGG <- as.data.frame(lizard)
#f   <- lizardAGG$Freq
#idx <- unlist(mapply(function(i, n) rep(i, n), 1:8, f))
#set.seed(0805)
#idx <- sample(idx)
#lizardRAW <- as.data.frame(lizardAGG[idx, 1:3])
#rownames(lizardRAW) <- 1:NROW(lizardRAW)
```

mathmark *Mathematics marks for students*

Description

The mathmark data frame has 88 rows and 5 columns.

Usage

```
mathmark
```

Format

This data frame contains the following columns: mechanics, vectors, algebra, analysis, statistics.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

David Edwards, An Introduction to Graphical Modelling, Second Edition, Springer Verlag, 2000

Examples

```
data(mathmark)
```

mildew *Mildew fungus*

Description

The data stem from a cross between two isolates of the barley powdery mildew fungus. For each offspring 6 binary characteristics, each corresponding to a single locus, were recorded. The object of the analysis is to determine the order of the loci along the chromosome.

Usage

```
mildew
```

Format

The format is: table [1:2, 1:2, 1:2, 1:2, 1:2, 1:2] 0 0 0 0 3 0 1 0 0 1 ... - attr(*, "dimnames")=List of 6 ..$ la10: chr [1:2] "1" "2" ..$ locc: chr [1:2] "1" "2" ..$ mp58: chr [1:2] "1" "2" ..$ c365: chr [1:2] "1" "2" ..$ p53a: chr [1:2] "1" "2" ..$ a367: chr [1:2] "1" "2"

References

Christiansen, S.K., Giese, H (1991) Genetic analysis of obligate barley powdery mildew fungus based on RFLP and virulence loci. Theor. Appl. Genet. 79:705-712

Examples

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

---

milkcomp *Milk composition data*

Description

Data from an experiment on composition of sow milk. Milk composition is measured on four occasions during lactation on a number of sows. The treatments are different types of fat added to the sows feed.

Usage

```
milkcomp
```

Format

A data frame with 214 observations on the following 7 variables.

`sow` a numeric vector

`lactime` a numeric vector

`treat` a factor with levels `a b c d e f g`

`fat` a numeric vector

`protein` a numeric vector

`dm` (dry matter) a numeric vector

`lactose` a numeric vector

Details

`a` is the control, i.e. no fat has been added.

`fat` + `protein` + `lactose` almost add up to `dm` (dry matter)

References

Charlotte Lauridsen and Viggo Danielsen (2004): Lactational dietary fat levels and sources influence milk composition and performance of sows and their progeny Livestock Production Science 91 (2004) 95-105

Examples

```
data(milkcomp)
## maybe str(milk) ; plot(milk) ...
```

Nutrimouse *The Nutrimouse Dataset*

Description

The data come from a study of the effects of five dietary regimens with different fatty acid compositions on liver lipids and hepatic gene expression in 40 mice.

Usage

```
Nutrimouse
```

Format

A data frame with 40 observations on the following 143 variables.

`genotype` a factor with levels `wt` `ppar`

`diet` a factor with levels `coc` `fish` `lin` `ref` `sun`

`X36b4` a numeric vector

`ACAT1` a numeric vector

`ACAT2` a numeric vector

`ACBP` a numeric vector

`ACC1` a numeric vector

`ACC2` a numeric vector

`ACOTH` a numeric vector

`ADISP` a numeric vector

`ADSS1` a numeric vector

`ALDH3` a numeric vector

`AM2R` a numeric vector

`AOX` a numeric vector

`BACT` a numeric vector

`BIEN` a numeric vector

`BSEP` a numeric vector

`Bcl.3` a numeric vector

`C16SR` a numeric vector

CACP a numeric vector

CAR1 a numeric vector

CBS a numeric vector

CIDEA a numeric vector

COX1 a numeric vector

COX2 a numeric vector

CPT2 a numeric vector

CYP24 a numeric vector

CYP26 a numeric vector

CYP27a1 a numeric vector

CYP27b1 a numeric vector

CYP2b10 a numeric vector

CYP2b13 a numeric vector

CYP2c29 a numeric vector

CYP3A11 a numeric vector

CYP4A10 a numeric vector

CYP4A14 a numeric vector

CYP7a a numeric vector

CYP8b1 a numeric vector

FAS a numeric vector

FAT a numeric vector

FDFT a numeric vector

FXR a numeric vector

G6PDH a numeric vector

G6Pase a numeric vector

GK a numeric vector

GS a numeric vector

GSTa a numeric vector

GSTmu a numeric vector

GSTpi2 a numeric vector

HMGCoAred a numeric vector

HPNCL a numeric vector

IL.2 a numeric vector

L.FABP a numeric vector

LCE a numeric vector

LDLr a numeric vector

LPK a numeric vector

LPL a numeric vector

LXRa a numeric vector

LXRb a numeric vector

Lpin a numeric vector

Lpin1 a numeric vector

Lpin2 a numeric vector

Lpin3 a numeric vector

M.CPT1 a numeric vector

MCAD a numeric vector

MDR1 a numeric vector

MDR2 a numeric vector

MRP6 a numeric vector

MS a numeric vector

MTHFR a numeric vector

NGFiB a numeric vector

NURR1 a numeric vector

Ntcp a numeric vector

OCTN2 a numeric vector

PAL a numeric vector

PDK4 a numeric vector

PECI a numeric vector

PLTP a numeric vector

PMDCI a numeric vector

PON a numeric vector

PPARa a numeric vector

PPARd a numeric vector

PPARg a numeric vector

PXR a numeric vector

Pex11a a numeric vector

RARa a numeric vector

RARb2 a numeric vector

RXRa a numeric vector

RXRb2 a numeric vector

RXRg1 a numeric vector

S14 a numeric vector

SHP1 a numeric vector

SIAT4c a numeric vector

SPI1.1 a numeric vector

SR.BI a numeric vector

THB a numeric vector

THIOL a numeric vector

TRa a numeric vector

TRb a numeric vector

Tpalpha a numeric vector

Tpbeta a numeric vector

UCP2 a numeric vector

UCP3 a numeric vector

VDR a numeric vector

VLDLr a numeric vector

Waf1 a numeric vector

ap2 a numeric vector

apoA.I a numeric vector

apoB a numeric vector

apoC3 a numeric vector

apoE a numeric vector

c.fos a numeric vector

cHMGCoAS a numeric vector

cMOAT a numeric vector

eif2g a numeric vector

hABC1 a numeric vector

i.BABP a numeric vector

i.BAT a numeric vector

i.FABP a numeric vector

i.NOS a numeric vector

mABC1 a numeric vector

mHMGCoAS a numeric vector

C14.0 a numeric vector

C16.0 a numeric vector

C18.0 a numeric vector

C16.1n.9 a numeric vector

C16.1n.7 a numeric vector

C18.1n.9 a numeric vector

C18.1n.7 a numeric vector

C20.1n.9 a numeric vector

`C20.3n.9` a numeric vector

`C18.2n.6` a numeric vector

`C18.3n.6` a numeric vector

`C20.2n.6` a numeric vector

`C20.3n.6` a numeric vector

`C20.4n.6` a numeric vector

`C22.4n.6` a numeric vector

`C22.5n.6` a numeric vector

`C18.3n.3` a numeric vector

`C20.3n.3` a numeric vector

`C20.5n.3` a numeric vector

`C22.5n.3` a numeric vector

`C22.6n.3` a numeric vector

Details

The data come from a study of the effects of five dietary regimens with different fatty acid compositions on liver lipids and hepatic gene expression in wild-type and PPAR-alpha-deficient mice (Martin et al., 2007).

There were 5 replicates per genotype and diet combination.

There are two design variables: (i) genotype, a factor with two levels: wild-type (wt) and PPAR-alpha-deficient (ppar), and (ii) diet, a factor with five levels. The oils used for experimental diet preparation were: corn and colza oils (50/50) for a reference diet (ref); hydrogenated coconut oil for a saturated fatty acid diet (coc); sunflower oil for an Omega6 fatty acid-rich diet (sun); linseed oil for an Omega3-rich diet (lin); and corn/colza/enriched (43/43/14) fish oils (fish).

There are 141 response variables: (i) the log-expression levels of 120 genes measured in liver cells, and (ii) the concentrations (in percentages) of 21 hepatic fatty acids measured by gas chromatography.

Source

The data were provided by Pascal Martin from the Toxicology and Pharmacology Laboratory, National Institute for Agronomic Research, French.

References

Martin, P. G. P., Guillou, H., Lasserre, F., D<e9>jean, S., Lan, A., Pascussi, J.-M., San Cristobal, M., Legrand, P., Besse, P. and Pineau, T. (2007). Novel aspects of PPARa-mediated regulation of lipid and xenobiotic metabolism revealed through a multrigenomic study. Hepatology 54, 767-777.

Examples

```
data(Nutrimouse)
```

old-array-operations *Array operations (2007)*

Description

Array operations; created to facilitate the gRain package in 2007. Now largely replaceable by other faster functions implemented in Rcpp.

Usage

```
tablePerm(tab, perm, resize = TRUE, keep.class = FALSE)

tableMult(tab1, tab2)

tableDiv(tab1, tab2)

tableOp(tab1, tab2, op = "*")

tableOp2(tab1, tab2, op = `*`, restore = FALSE)

tableSlice(tab, margin, level, impose)

tableSlicePrim(tab, mar.idx, lev.idx)

tableMargin(tab, margin, keep.class = FALSE)

tableGetSliceIndex(tab, margin, level, complement = FALSE)

tableSetSliceValue(tab, margin, level, complement = FALSE, value = 0)
```

Arguments

tab, tab1, tab2	Arrays with named dimnames.
perm	A permutation; either indices or names.
resize	A flag indicating whether the vector should be resized as well as having its elements reordered (default TRUE).
keep.class	Obsolete argument.
op	The operation; choices are "*", "/", "+", "-".
restore	Not so clear anymore.
margin	Index or name of margin.
level	Corresponding level of margin.
impose	Value to be imposed.
mar.idx	Index of margin

lev.idx	Index of level
complement	Should values be set for the complement?
value	Which value should be set

---

old-parray *Representation of and operations on multidimensional arrays*

Description

General representation of multidimensional arrays (with named dimnames, also called named arrays.)

Usage

```
parray(varNames, levels, values = 1, normalize = "none", smooth = 0)

as.parray(values, normalize = "none", smooth = 0)
```

Arguments

varNames	Names of variables defining table; can be a right hand sided formula.
levels	Either 1) a vector with number of levels of the factors in varNames or 2) a list with specification of the levels of the factors in varNames. See 'examples' below.
values	Values to go into the array
normalize	Either "none", "first" or "all". Should result be normalized, see 'Details' below.
smooth	Should values be smoothed, see 'Details' below.

Details

A named array object represents a table defined by a set of variables and their levels, together with the values of the table. E.g. f(a,b,c) can be a table with a,b,c representing levels of binary variable

If `normalize="first"` then for each configuration of all other variables than the first, the probabilities are normalized to sum to one. Thus f(a,b,c) becomes a conditional probability table of the form p(a|b,c). If `normalize="all"` then the sum over all entries of f(a,b,c) is one.

If `smooth` is positive then `smooth` is added to `values` before normalization takes place.

Value

A a named array.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

is.named.array, ar_marg

Examples

```
t1 <- parray(c("gender","answer"), list(c('male','female'),c('yes','no')), values=1:4)
t1 <- parray(~gender:answer, list(c('male','female'),c('yes','no')), values=1:4)
t1 <- parray(~gender:answer, c(2,2), values=1:4)

t2 <- parray(c("answer","category"), list(c('yes','no'),c(1,2)), values=1:4+10)
t3 <- parray(c("category","foo"), c(2,2), values=1:4+100)

varNames(t1)
nLevels(t1)
valueLabels(t1)


## Create 1-dimensional vector with dim and dimnames
x1 <- 1:5
as.parray(x1)
x2 <- parray("x", levels=length(x1), values=x1)
dim(x2)
dimnames(x2)

## Matrix
x1 <- matrix(1:6, nrow=2)
as.parray(x1)
parray(~a:b, levels=dim(x1), values=x1)

## Extract parrays from data
## 1) a dataframe
data(cad1)
data2parray(cad1, ~Sex:AngPec:AMI)
data2parray(cad1, c("Sex","AngPec","AMI"))
data2parray(cad1, c(1,2,3))
## 2) a table
data2parray(UCBAdmissions,c(1,2), normalize="first")
```

---

rats *Weightloss of rats*

Description

An artificial dataset. 24 rats (12 female, 12 male) have been randomized to use one of three drugs (products for loosing weight). The weightloss for each rat is noted after one and two weeks.

Usage

```
rats
```

Format

A dataframe with 4 variables. Sex: "M" (male), "F" (female). Drug: "D1", "D2", "D3" (three types). W1 weightloss, week one. W2 weightloss, week 2.

References

Morrison, D.F. (1976). Multivariate Statistical Methods. McGraw-Hill, USA.

Edwards, D. (1995). Introduction to Graphical Modelling, Springer-Verlag. New York.

---

reinis *Risk factors for coronary heart disease.*

Description

Data collected at the beginning of a 15 year follow-up study of probable risk factors for coronary thrombosis. Data are from all men employed in a car factory.

Usage

```
reinis
```

Format

A table with 6 discrete variables. A: smoking, B: strenous mental work, D: strenuous physical work, E: systolic blood pressure, F: ratio of lipoproteins, G: Family anamnesis of coronary heart disease.

References

Edwards and Havranek (1985): A fast procedure for model search in multidimensional contingency tables. Biometrika, 72: 339-351.

Reinis et al (1981): Prognostic significance of the risk profile in the prevention of coronary heart disease. Bratis. lek. Listy. 76: 137-150.

Setoperations *Set operations*

Description

Miscellaneous set operations.

Usage

```
is.subsetof(x, set)
is.insetlist(x, setlist, index=FALSE)
removeRedundant(setlist, maximal = TRUE, index = FALSE)
maximalSets(setlist,  index = FALSE)
minimalSets(setlist,  index = FALSE)
```

Arguments

x, set	Vectors representing sets
setlist	List of vectors (representing a set of subsets)
maximal	Logical; see section 'Details' for a description.
index	Logical; should indices (in setlist) be returned or a set of subsets.

Details

'setlist' is a list of vectors representing a set of subsets; i.e. V1,...VQ where Vk is a subset of some base set V.

`is.insetlist`: Checks if the set x is in one of the Vk's.

`removeRedundant`: Returns those Vk which are not contained in other subsets; i.e. gives the maximal sets. If maximal is FALSE then returns the minimal sets; i.e. Vk is returned if Vk is contained in one of the other sets Vl and there are no set Vn contained in Vk.

Notice that the comparisons are made by turning the elements into characters and then comparing these. Hence 1 is identical to "1".

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
is.subsetof(c(1,2),c(1,2,3))
is.subsetof(c(1,2,3), c(1,2))

l <- list(c(1,2),c(1,2,3),c(2,4),c(5,6), 5)

#subsetofList(c(1,2), l)
```

```
#subsetofList(c(1,2,3,4), l)

removeRedundant(l)
removeRedundant(l, maximal=FALSE)

is.insetlist (c(2,4), l)
is.insetlist (c(2,8), l)
```

---

simulateArray *Simulate data from array.*

Description

Simulate data (slice of) an array.

Usage

```
simulateArray(x, nsim = 1, margin, value.margin)
```

Arguments

x	An array
nsim	Number of cases to simulate
margin, value.margin	Specification of slice of array to simulate from

Value

A matrix

Note

The current implementation is fragile in the sense that it is not checked that the input argument x is an array.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
## 2x2 array
x <- parray(c("a","b"), levels=c(2,2), values=1:4)

## Simulate from entire array
s <-simulateArray(x,1000)
```

```
xtabs(~., as.data.frame(s))

## Simulate from slice defined by that dimension 1 is fixed at level 2
s <-simulateArray(x, 6000, 1, 2)
xtabs(~., as.data.frame(s))

## 2x2x2 array
x <- parray(c("a", "b", "c"), levels=c(2, 2, 2), values=1:8)

## Simulate from entire array
s <-simulateArray(x, 36000)
xtabs(~., as.data.frame(s))

## Simulate from slice defined by that dimension 3 is fixed at level 1
s <-simulateArray(x, 10000, 3, 1)
xtabs(~., as.data.frame(s))
```

`wine` *Chemical composition of wine*

Description

Using chemical analysis determine the origin of wines

Usage

```
wine
```

Format

A data frame with 178 observations on the following 14 variables.

`Cult` a factor with levels `v1` `v2` `v3`: 3 different graph varieties

`Alch` Alcohol

`Mlca` Malic acid

`Ash` Ash

`Aloa` Alcalinity of ash

`Mgns` Magnesium

`Ttlp` Total phenols

`Flvn` Flavanoids

`Nnfp` Nonflavanoid phenols

`Prnt` Proanthocyanins

`Clri` Color intensity

`Hue` Hue

`Oodw` OD280/OD315 of diluted wines

`Prln` Proline

Details

Data comes from the UCI Machine Learning Repository. The grape variety `Cult` is the class identifier.

Source

Frank, A. & Asuncion, A. (2010). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

References

See references at http://archive.ics.uci.edu/ml/datasets/Wine

Examples

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

Index

∗Topic **datasets**
ashtrees, 14
BodyFat, 15
breastcancer, 16
cad, 17
carcass, 18
chestSim, 19
dietox, 22
dumping, 23
lizard, 55
mathmark, 56
mildew, 56
milkcomp, 57
Nutrimouse, 58
rats, 65
reinis, 66
wine, 69
∗Topic **graphics**
graph-iplot, 30
∗Topic **graphs**
gRbase, 52
∗Topic **models**
graph-gcproperties, 28
gRbase, 52
∗Topic **multivariate**
gRbase, 52
∗Topic **utilities**
array-create, 4
array-distribution, 7
array-operations, 9
combnPrim, 20
compareModels, 21
cov2pcor, 22
graph-cliques, 24
graph-create, 26
graph-gcproperties, 28
graph-is, 31
graph-mcs, 34
graph-minimaltriang, 35
graph-moralize, 37
graph-mpd, 38
graph-randomdag, 40
graph-rip, 41
graph-toposort, 43
graph-triang, 44
graph-triangulate, 46
graph-xxx2yyy, 50
gRbase-generics, 53
gRbase-utilities, 54
old-parray, 64
Setoperations, 67
simulateArray, 68
%a*% *(array-algebra)*, 3
%a+% *(array-algebra)*, 3
%a-% *(array-algebra)*, 3
%a/0% *(array-algebra)*, 3
%a/% *(array-algebra)*, 3
%a==% *(array-operations)*, 9
%a_% *(array-operations)*, 9
%a^% *(array-operations)*, 9

adjList2adjMAT *(graph-xxx2yyy)*, 50
adjList2M *(graph-xxx2yyy)*, 50
ancestors *(graph-query)*, 39
ancestralGraph *(graph-query)*, 39
ancestralSet *(graph-query)*, 39
aperm, *10*
ar_add, *5*, *13*
ar_add *(array-algebra)*, 3
ar_align *(array-operations)*, 9
ar_dist *(array-distribution)*, 7
ar_div, *13*
ar_div *(array-algebra)*, 3
ar_div0 *(array-algebra)*, 3
ar_equal *(array-operations)*, 9
ar_expand *(array-operations)*, 9
ar_marg, *7*, *13*, *65*
ar_marg *(array-operations)*, 9
ar_mult, *13*

ar_mult (array-algebra), 3
ar_new, 7
ar_new (array-create), 4
ar_normalize (array-normalize), 8
ar_perm, 5, 10, 13
ar_perm (array-operations), 9
ar_prod, 5, 13
ar_prod (array-algebra), 3
ar_prod_list (array-operations), 9
ar_slice, 7, 10
ar_slice (array-slice), 12
ar_slice_entries, 10
ar_slice_entries (array-slice), 12
ar_slice_mult (array-slice), 12
ar_slice_prim (array-slice), 12
ar_subt, 13
ar_subt (array-algebra), 3
ar_sum, 13
ar_sum (array-algebra), 3
ar_sum_list (array-operations), 9
array-algebra, 3
array-create, 4
array-dimnames, 6
array-distribution, 7
array-normalize, 8
array-operations, 9
array-properties, 11
array-slice, 12
as.adjMAT (graph-xxx2yyy), 50
as.parray (old-parray), 64
ashtrees, 14

BodyFat, 15
breastcancer, 16

cad, 17
cad1 (cad), 17
cad2 (cad), 17
carcass, 18
carcassall (carcass), 18
chestSim, 19
chestSim1000 (chestSim), 19
chestSim10000 (chestSim), 19
chestSim100000 (chestSim), 19
chestSim500 (chestSim), 19
chestSim50000 (chestSim), 19
children (graph-query), 39
closure (graph-query), 39
coerceGraph (graph-coerce), 25
colwiseProd (gRbase-utilities), 54
combn, 20
combnPrim, 20
compareModels, 21
compile (gRbase-generics), 53
compile.grain, 54
conc2pcor (cov2pcor), 22
cov2pcor, 22

dag, 25, 32, 35, 38, 42, 44, 46, 47, 49, 51
dag (graph-create), 26
dagList (graph-create), 26
dagList2dgCMatrix (graph-create), 26
dagList2graphNEL (graph-xxx2yyy), 50
dagList2M (graph-xxx2yyy), 50
dagList2matrix (graph-create), 26
dagList2tfM (graph-xxx2yyy), 50
data2parray (old-parray), 64
df2xtabs (array-create), 4
dietox, 22
dimnames_match (array-dimnames), 6
dimnames_match_ (array-dimnames), 6
dumping, 23

edgeList (graph-edgeList), 27
edgeListMAT (graph-edgeList), 27

fit (gRbase-generics), 53

get_subset_ (gRbase-utilities), 54
get_superset_ (gRbase-utilities), 54
getCliques (graph-cliques), 24
glist2adjMAT (graph-xxx2yyy), 50
graph-cliques, 24
graph-coerce, 25
graph-create, 26
graph-edgeList, 27
graph-gcproperties, 28
graph-iplot, 30
graph-is, 31
graph-mcs, 33
graph-minimaltriang, 35
graph-moralize, 37
graph-mpd, 38
graph-query, 39
graph-randomdag, 40
graph-rip, 41
graph-toposort, 43
graph-triang, 44

graph-triangulate, 46
graph-vpar, 48
graph-xxx2yyy, 50
graphNEL2adjMAT (graph-xxx2yyy), 50
graphNEL2dgCMatrix (graph-xxx2yyy), 50
graphNEL2ftM (graph-xxx2yyy), 50
graphNEL2igraph (graph-xxx2yyy), 50
graphNEL2M (graph-xxx2yyy), 50
graphNEL2MAT (graph-xxx2yyy), 50
graphNEL2matrix (graph-xxx2yyy), 50
graphNEL2tfM (graph-xxx2yyy), 50
gRbase, 52
gRbase-generics, 53
gRbase-utilities, 54

iplot (graph-iplot), 30
is.adjMAT (graph-is), 31
is.complete (graph-query), 39
is.DAG (graph-is), 31
is.DAGMAT (graph-is), 31
is.decomposition (graph-query), 39
is.DG (graph-is), 31
is.DGMAT (graph-is), 31
is.insetlist (Setoperations), 67
is.named.array, *6*, *65*
is.named.array (array-properties), 11
is.simplicial (graph-query), 39
is.subsetof (Setoperations), 67
is.TUG (graph-is), 31
is.TUGMAT (graph-is), 31
is.UG (graph-is), 31
is.UGMAT (graph-is), 31
is_dimnames_ (array-properties), 11
is_named_array_ (array-properties), 11
is_number_vector_ (array-properties), 11
is_subsetof_ (gRbase-utilities), 54
isDecomposable (graph-gcproperties), 28
isGraphical (graph-gcproperties), 28
isin (Setoperations), 67

jTree, *35*, *38*
jTree (graph-rip), 41
jTreeMAT (graph-rip), 41
junctionTree (graph-rip), 41
junctionTreeMAT (graph-rip), 41

lizard, 55
lizardAGG (lizard), 55
lizardRAW (lizard), 55

M2adjList (graph-xxx2yyy), 50
M2dagList (graph-xxx2yyy), 50
M2graphNEL (graph-xxx2yyy), 50
M2igraph (graph-xxx2yyy), 50
M2ugList (graph-xxx2yyy), 50
MAT2dgCMatrix (graph-xxx2yyy), 50
MAT2matrix (graph-xxx2yyy), 50
math (mathmark), 56
mathmark, 56
maxCliqueMAT (graph-cliques), 24
maximalSets (Setoperations), 67
mcs, *25*, *29*, *38*, *39*, *42*, *46*, *47*
mcs (graph-mcs), 34
mcsmarked (graph-mcs), 34
mcsmarkedMAT (graph-mcs), 34
mcsMAT, *25*, *39*, *46*, *47*
mcsMAT (graph-mcs), 34
mildew, 56
milkcomp, 57
milkcomp1 (milkcomp), 57
minimalSets (Setoperations), 67
minimalTriang, *39*
minimalTriang (graph-minimaltriang), 35
minimalTriangMAT, *39*
minimalTriangMAT (graph-minimaltriang), 35
moralize, *25*, *35*, *42*, *46*, *47*
moralize (graph-moralize), 37
moralizeMAT, *25*, *46*, *47*
moralizeMAT (graph-moralize), 37
mpd, *36*
mpd (graph-mpd), 38
mpdMAT (graph-mpd), 38

newar (array-create), 4
nonEdgeList (graph-edgeList), 27
nonEdgeListMAT (graph-edgeList), 27
Nutrimouse, 58

old-array-operations, 63
old-parray, 64

parents (graph-query), 39
parray (old-parray), 64
propagate (gRbase-generics), 53
propagate.grain, *54*

querygraph (graph-query), 39

random_dag (graph-randomdag), 40

rats, 65
reinis, 66
removeRedundant (Setoperations), 67
rip, *25*, *29*, *35*, *36*, *38*, *39*, *46*, *47*
rip (graph-rip), 41
ripMAT, *25*, *39*, *46*, *47*
ripMAT (graph-rip), 41

Setoperations, 67
simplicialNodes (graph-query), 39
simulateArray, 68
stepwise (gRbase-generics), 53
subsetof (Setoperations), 67

tab (array-create), 4
tabAdd (array-algebra), 3
tabAdd__ (array-operations), 9
tabAlign (array-operations), 9
tabAlign__ (array-operations), 9
tabCondProb (array-operations), 9
tabDist (array-distribution), 7
tabDiv (array-algebra), 3
tabDiv0 (array-algebra), 3
tabDiv0__ (array-operations), 9
tabDiv__ (array-operations), 9
tabEqual (array-operations), 9
tabEqual__ (array-operations), 9
tabExpand (array-operations), 9
tabExpand__ (array-operations), 9
tabExt (array-operations), 9
tableDiv (old-array-operations), 63
tableGetSliceIndex
(old-array-operations), 63
tableMargin (old-array-operations), 63
tableMult (old-array-operations), 63
tableOp (old-array-operations), 63
tableOp2 (old-array-operations), 63
tablePerm (old-array-operations), 63
tableSetSliceValue
(old-array-operations), 63
tableSlice (old-array-operations), 63
tableSlicePrim (old-array-operations),
63
tabListAdd (array-operations), 9
tabListAdd__ (array-operations), 9
tabListMult (array-operations), 9
tabListMult__ (array-operations), 9
tabMarg (array-operations), 9
tabMarg__ (array-operations), 9
tabMult (array-algebra), 3
tabMult__ (array-operations), 9
tabNormalize (array-normalize), 8
tabOp__ (array-operations), 9
tabPerm (array-operations), 9
tabPerm__ (array-operations), 9
tabProd (array-algebra), 3
tabSlice (array-operations), 9
tabSlice2 (array-operations), 9
tabSlice2Entries (array-operations), 9
tabSlice2Entries_ (array-operations), 9
tabSliceMult (array-operations), 9
tabSlicePrim (array-operations), 9
tabSubt (array-algebra), 3
tabSubt__ (array-operations), 9
tabSum (array-algebra), 3
topoSort (graph-toposort), 43
topoSort_vparList (graph-toposort), 43
topoSortMAT (graph-toposort), 43
triang (graph-triang), 44
triang_elo (graph-triang), 44
triang_eloMAT (graph-triang), 44
triang_eloMAT_ (graph-triang), 44
triang_mcwh (graph-triang), 44
triangulate, *36*, *39*, *42*
triangulate (graph-triangulate), 46
triangulateMAT, *39*
triangulateMAT (graph-triangulate), 46

ug, *25*, *32*, *35*, *38*, *42*, *44*, *46*, *47*, *49*, *51*
ug (graph-create), 26
ug2dag (graph-xxx2yyy), 50
ugList (graph-create), 26
ugList2dgCMatrix (graph-create), 26
ugList2graphNEL (graph-xxx2yyy), 50
ugList2M (graph-xxx2yyy), 50
ugList2matrix (graph-create), 26

vchi (graph-vpar), 48
vchiMAT (graph-vpar), 48
vpaL2tfM (graph-xxx2yyy), 50
vpaList2adjMAT (graph-xxx2yyy), 50
vpar (graph-vpar), 48
vparMAT (graph-vpar), 48

wine, 69