Package ‘gRim’

October 16, 2022

Version 0.2.10

Title Graphical Interaction Models

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

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

Description Provides the following types of models: Models for contingency tables (i.e. log-linear models) Graphical Gaussian models for multivariate normal data (i.e. covariance selection models) Mixed interaction models. Documentation about 'gRim' is provided by vignettes included in this package and the book by Højsgaard, Edwards and Lauritzen (2012, <doi:10.1007/978-1-4614-2299-0>); see 'citation(``gRim'')' for details.

License GPL (>= 2)

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

Encoding UTF-8

Depends R (>= 3.6.0), methods, gRbase (>= 1.8.6)

Suggests testthat (>= 2.1.0)

Imports graph, Rgraphviz, igraph, stats4, gRain (>= 1.3.10), magrittr, Rcpp (>= 0.11.1)

ByteCompile yes

biocViews

LinkingTo Rcpp (>= 0.11.1), RcppArmadillo, RcppEigen, gRbase (>= 1.8.6)

RoxygenNote 7.2.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-10-15 22:22:35 UTC

R topics documented:

`cg-stats` *Mean, covariance and counts for grouped data (statistics for conditional Gaussian distribution).*

Description

`CGstats` provides what corresponds to calling `cow.wt` on different strata of data where the strata are defined by the combinations of factors in data.

Usage

```
CGstats(object, varnames = NULL, homogeneous = TRUE, simplify = TRUE)
```

Arguments

`object`	A dataframe.
`varnames`	Names of variables to be used.
`homogeneous`	Logical; if TRUE a common covariance matrix is reported.
`simplify`	Logical; if TRUE the result will be presented in a simpler form.

Value

A list whose form depends on the type of input data and the varnames.

Author(s)

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

See Also

`cov.wt`

Examples

```
data(milkcomp)
# milkcomp <- subset(milkcomp, (treat %in% c("a", "b")) & (lactime %in% c("t1", "t2")))
# milkcomp <- milkcomp[,-1]
# milkcomp$treat  <- factor(milkcomp$treat)
# milkcomp$lactime  <- factor(milkcomp$lactime)

CGstats(milkcomp)
CGstats(milkcomp, c(1, 2))
CGstats(milkcomp, c("lactime", "treat"))
CGstats(milkcomp, c(3, 4))
CGstats(milkcomp, c("fat", "protein"))

CGstats(milkcomp, c(2, 3, 4), simplify=FALSE)
CGstats(milkcomp, c(2, 3, 4), homogeneous=FALSE)
CGstats(milkcomp, c(2, 3, 4), simplify=FALSE, homogeneous=FALSE)
```

`citest-array` *Test for conditional independence in a contingency table*

Description

Test for conditional independence in a contingency table represented as an array.

Usage

```
ciTest_table(
  x,
  set = NULL,
  statistic = "dev",
  method = "chisq",
  adjust.df = TRUE,
  slice.info = TRUE,
```

```
  L = 20,
  B = 200,
  ...
)
```

Arguments

`x`	An array of counts with named dimnames.
`set`	A specification of the test to be made. The tests are of the form u and v are independent condionally on S where u and v are variables and S is a set of variables. See 'details' for details about specification of `set`.
`statistic`	Possible choices of the test statistic are `"dev"` for deviance and `"X2"` for Pearsons X2 statistic.
`method`	Method of evaluating the test statistic. Possible choices are `"chisq"`, `"mc"` (for Monte Carlo) and `"smc"` for sequential Monte Carlo.
`adjust.df`	Logical. Should degrees of freedom be adjusted for sparsity?
`slice.info`	Logical. Should slice info be stored in the output?
`L`	Number of extreme cases as stop criterion if method is `"smc"` (sequential Monte Carlo test); ignored otherwise.
`B`	Number (maximum) of simulations to make if method is `"mc"` or `"smc"` (Monte Carlo test or sequential Monte Carlo test); ignored otherwise.
`...`	Additional arguments.

Details

`set` can be 1) a vector or 2) a right-hand sided formula in which variables are separated by '+'. In either case, it is tested if the first two variables in the `set` are conditionally independent given the remaining variables in `set`. (Notice an abuse of the '+' operator in the right-hand sided formula: The order of the variables does matter.)

If `set` is `NULL` then it is tested whether the first two variables are conditionally independent given the remaining variables.

Value

An object of class `citest` (which is a list).

Author(s)

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

See Also

`ciTest`, `ciTest_df`, `ciTest_mvn`, `chisq.test`

Examples

```
data(lizard)

## lizard is has named dimnames
names( dimnames( lizard ))
## checked with
is.named.array( lizard )

## Testing for conditional independence:
# the following are all equivalent:
ciTest(lizard, set=~diam + height + species)
# ciTest(lizard, set=c("diam", "height", "species"))
# ciTest(lizard, set=1:3)
# ciTest(lizard)
# (The latter because the names in lizard are as given above.)

## Testing for marginal independence
ciTest(lizard, set=~diam + height)
ciTest(lizard, set=1:2)

## Getting slice information:
ciTest(lizard, set=c("diam", "height", "species"), slice.info=TRUE)$slice

## Do Monte Carlo test instead of usual likelihood ratio test. Different
# options:

# 1) Do B*10 simulations divided equally over each slice:
ciTest(lizard, set=c("diam", "height", "species"), method="mc", B=400)
# 2) Do at most B*10 simulations divided equally over each slice, but stop
# when at most L extreme values are found
ciTest(lizard, set=c("diam", "height", "species"), method="smc", B=400)
```

`citest-df` *Test for conditional independence in a dataframe*

Description

Test for conditional independence in a dataframe.

Usage

```
ciTest_df(x, set = NULL, ...)
```

Arguments

`x`	A dataframe.
`set`	A specification of the test to be made. The tests are of the form u and v are independent condionally on S where u and v are variables and S is a set of variables. See 'details' for details about specification of `set`.

`...`	Additional arguments.

Details

- `set` can be 1) a vector or 2) a right-hand sided formula in which variables are separated by '+'. In either case, it is tested if the first two variables in the `set` are conditionally independent given the remaining variables in `set`. (Notice an abuse of the '+' operator in the right-hand sided formula: The order of the variables does matter.)
- If `set` is `NULL` then it is tested whether the first two variables are conditionally independent given the remaining variables.
- If `set` consists only of factors then `x[,set]` is converted to a contingency table and the test is made in this table using `ciTest_table()`.
- If `set` consists only of numeric values and integers then `x[,set]` is converted to a list with components `cov` and `n.obs` by calling `cov.wt(x[,set], method='ML')`. This list is then passed on to `ciTest_mvn()` which makes the test.

Value

An object of class `citest` (which is a list).

Author(s)

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

See Also

`ciTest`, `ciTest_table`, `ciTest_mvn`, `chisq.test`

Examples

```
data(milkcomp1)
ciTest(milkcomp1, set=~tre + fat + pro)
ciTest_df(milkcomp1, set=~tre + fat + pro)
```

`citest-generic` *Generic function for conditional independence test*

Description

Generic function for conditional independence test. Specializes to specific types of data.

Usage

```
ciTest(x, set = NULL, ...)
```

Arguments

`x`	An object for which a test for conditional independence is to be made. See 'details' for valid types of `x`.
`set`	A specification of the test to be made. The tests are of the form u and v are independent condionally on S where u and v are variables and S is a set of variables. See 'details' for details about specification of `set`.
`...`	Additional arguments to be passed on to other methods.

Details

`x` can be

1. a table (an array). In this case `ciTest_table` is called.
2. a dataframe whose columns are numerics and factors. In this case `ciTest_df` is called.
3. a list with components `cov` and `n.obs`. In this case `ciTest_mvn` is called.

`set` can be

1. a vector,
2. a right-hand sided formula in which variables are separated by '+'.

In either case, it is tested if the first two variables in the `set` are conditionally independent given the remaining variables in `set`. (Notice an abuse of the '+' operator in the right-hand sided formula: The order of the variables does matter.)

Value

An object of class `citest` (which is a list).

Author(s)

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

See Also

`ciTest_table`, `ciTest_df`, `ciTest_mvn`, `chisq.test`

Examples

```
## contingency table:
data(reinis)
## dataframe with only numeric variables:
data(carcass)
## dataframe with numeric variables and factors:
data(milkcomp1)

ciTest(cov.wt(carcass, method='ML'), set=~Fat11 + Meat11 + Fat12)
ciTest(reinis, set=~smo + phy + sys)
ciTest(milkcomp1, set=~tre + fat + pro)
```

`citest-mvn` *Test for conditional independence in the multivariate normal distribution*

Description

Test for conditional independence in the multivariate normal distribution.

Usage

```
ciTest_mvn(x, set = NULL, statistic = "DEV", ...)
```

Arguments

`x`	A list with elements `cov` and `n.obs` (such as returned from calling `cov.wt()` on a dataframe. See examples below.)
`set`	A specification of the test to be made. The tests are of the form u and v are independent condionally on S where u and v are variables and S is a set of variables. See 'details' for details about specification of `set`.
`statistic`	The test statistic to be used, valid choices are `"DEV"` and `"F"`.
`...`	Additional arguments

Details

`set` can be 1) a vector or 2) a right-hand sided formula in which variables are separated by '+'. In either case, it is tested if the first two variables in the `set` are conditionally independent given the remaining variables in `set`. (Notice an abuse of the '+' operator in the right-hand sided formula: The order of the variables does matter.)

If `set` is `NULL` then it is tested whether the first two variables are conditionally independent given the remaining variables.

`x` must be a list with components `cov` and `n.obs` such as returned by calling `cov.wt( , method='ML')` on a dataframe.

Value

An object of class `citest` (which is a list).

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

`ciTest`, `ciTest_table`, `ciTest_df`, `ciTest_mvn`, `chisq.test`

Examples

```
data(carcass)
ciTest(cov.wt(carcass, method='ML'), set=~Fat11 + Meat11 + Fat12)
ciTest_mvn(cov.wt(carcass, method='ML'), set=~Fat11 + Meat11 + Fat12)
```

`citest-ordinal` *A function to compute Monte Carlo and asymptotic tests of conditional independence for ordinal and/or nominal variables.*

Description

The function computes tests of independence of two variables, say u and v, given a set of variables, say S. The deviance, Wilcoxon, Kruskal-Wallis and Jonkheere-Terpstra tests are supported. Asymptotic and Monte Carlo p-values are computed.

Usage

```
ciTest_ordinal(x, set = NULL, statistic = "dev", N = 0, ...)
```

Arguments

`x`	A dataframe or table.
`set`	The variable set (u,v,S), given either as an integer vector of the column numbers of a dataframe or dimension numbers of a table, or as a character vector with the corresponding variable or dimension names.
`statistic`	Either "deviance", "wilcoxon", "kruskal" or "jt".
`N`	The number of Monte Carlo samples. If N<=0 then Monte Carlo p-values are not computed.
`...`	Additional arguments, currently not used

Details

The deviance test is appropriate when u and v are nominal; Wilcoxon, when u is binary and v is ordinal; Kruskal-Wallis, when u is nominal and v is ordinal; Jonckheere-Terpstra, when both u and v are ordinal.

Value

A list including the test statistic, the asymptotic p-value and, when computed, the Monte Carlo p-value.

`P`	Asymptotic p-value
`montecarlo.P`	Monte Carlo p-value

Author(s)

Flaminia Musella, David Edwards, Søren Højsgaard, <sorenh@math.aau.dk>

References

See Edwards D. (2000), "Introduction to Graphical Modelling", 2nd ed., Springer-Verlag, pp. 130-153.

See Also

`ciTest_table`, `ciTest`

Examples

```
library(gRim)
data(dumping, package="gRbase")

ciTest_ordinal(dumping, c(2,1,3), stat="jt", N=1000)
ciTest_ordinal(dumping, c("Operation", "Symptom", "Centre"), stat="jt", N=1000)
ciTest_ordinal(dumping, ~Operation + Symptom + Centre, stat="jt", N=1000)

data(reinis)
ciTest_ordinal(reinis, c(1,3,4:6), N=1000)

# If data is a dataframe
dd     <- as.data.frame(dumping)
ncells <- prod(dim(dumping))
ff     <- dd$Freq
idx    <- unlist(mapply(function(i,n) rep(i,n),1:ncells,ff))
dumpDF <- dd[idx, 1:3]
rownames(dumpDF) <- 1:NROW(dumpDF)

ciTest_ordinal(dumpDF, c(2,1,3), stat="jt", N=1000)
ciTest_ordinal(dumpDF, c("Operation","Symptom","Centre"), stat="jt", N=1000)
ciTest_ordinal(dumpDF, ~ Operation + Symptom + Centre, stat="jt", N=1000)
```

`cmod` *Graphical Gaussian model*

Description

Specification of graphical Gaussian model. The 'c' in the name `cmod` refers to that it is a (graphical) model for 'c'ontinuous variables

Usage

```
cmod(formula, data, marginal = NULL, fit = TRUE, details = 0)
```

Arguments

`formula`	Model specification in one of the following forms: 1) a right-hand sided formula, 2) as a list of generators. Notice that there are certain model specification shortcuts, see Section 'details' below.
`data`	Data in one of the following forms: 1) A dataframe or 2) a list with elements `cov` and `n.obs` (such as returned by the `cov.wt()` function.)
`marginal`	Should only a subset of the variables be used in connection with the model specification shortcuts.
`fit`	Should the model be fitted.
`details`	Control the amount of output; for debugging purposes.

Details

The independence model can be specified as `~.^1` and the saturated model as `~.^.`. The `marginal` argument can be used for specifying the independence or saturated models for only a subset of the variables.

Value

An object of class `cModel` (a list)

Author(s)

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

See Also

`dmod`, `mmod`, `ggmfit`

Examples

```
## Graphical Gaussian model
data(carcass)
cm1 <- cmod(~ .^., data=carcass)

## Stepwise selection based on BIC
cm2 <- backward(cm1, k=log(nrow(carcass)))

## Stepwise selection with fixed edges
cm3 <- backward(cm1, k=log(nrow(carcass)),
 fixin=matrix(c("LeanMeat", "Meat11", "Meat12", "Meat13",
                "LeanMeat", "Fat11", "Fat12", "Fat13"),
                 ncol=2))
```

`getEdges` *Find edges in a graph or edges not in an undirected graph.*

Description

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

Usage

```
getEdges(object, type = "unrestricted", ingraph = TRUE, discrete = NULL, ...)
```

Arguments

`object`	An object representing a graph; either a generator list, a graphNEL object or an adjacency matrix.
`type`	Either "unrestricted" or "decomposable"
`ingraph`	If TRUE the result is the edges in the graph; if FALSE the result is the edges not in the graph.
`discrete`	This argument is relevant only if `object` specifies a marked graph in which some vertices represent discrete variables and some represent continuous variables.
`...`	Additional arguments; currently not used.

Details

When `ingraph=TRUE`: If type="decomposable" then `getEdges()` returns those edges e for which the graph with e removed is decomposable.

When `ingraph=FALSE`: Likewise, if type="decomposable" then `getEdges()` returns those edges e for which the graph with e added is decomposable.

The functions `getInEdges()` and `getInEdges()` are just wrappers for calls to `getEdges()`.

The workhorses are `getInEdgesMAT()` and `getOutEdgesMAT()` and these work on adjacency matrices.

Regarding the argument `discrete`, please see the documentation of `mcs_marked`.

Value

A p * 2 matrix with edges.

Note

These functions work on undirected graphs. The behaviour is undocumented for directed graphs.

Author(s)

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

See Also

`edgeList`, `nonEdgeList`.

Examples

```
gg     <- ug(~a:b:d + a:c:d + c:e)
glist  <- getCliques(gg)
adjmat <- as.adjMAT(gg)

#### On a glist
getEdges(glist)
getEdges(glist, type="decomposable")
# Deleting (a,d) would create a 4-cycle

getEdges(glist, ingraph=FALSE)
getEdges(glist, type="decomposable", ingraph=FALSE)
# Adding (e,b) would create a 4-cycle

#### On a graphNEL
getEdges(gg)
getEdges(gg, type="decomposable")
# Deleting (a,d) would create a 4-cycle

getEdges(gg, ingraph=FALSE)
getEdges(gg, type="decomposable", ingraph=FALSE)
# Adding (e,b) would create a 4-cycle

#### On an adjacency matrix
getEdges(adjmat)
getEdges(adjmat, type="decomposable")
# Deleting (a,d) would create a 4-cycle

getEdges(adjmat, ingraph=FALSE)
getEdges(adjmat, type="decomposable", ingraph=FALSE)
# Adding (e,b) would create a 4-cycle


## Marked graphs; vertices a,b are discrete; c,d are continuous
UG <- ug(~a:b:c + b:c:d)
disc <- c("a","b")
getEdges(UG)
getEdges(UG, discrete=disc)
## Above: same results; there are 5 edges in the graph

getEdges(UG, type="decomposable")
## Above: 4 edges can be removed and will give a decomposable graph
##(only removing the edge (b,c) would give a non-decomposable model)

getEdges(UG, type="decomposable", discrete=c("a","b"))
## Above: 3 edges can be removed and will give a strongly decomposable
## graph. Removing (b,c) would create a 4--cycle and removing (a,b)
```

```
## would create a forbidden path; a path with only continuous vertices
## between two discrete vertices.
```

`ggmfit` *Iterative proportional fitting of graphical Gaussian model*

Description

Fit graphical Gaussian model by iterative proportional fitting.

Usage

```
ggmfit(
  S,
  n.obs,
  glist,
  start = NULL,
  eps = 1e-12,
  iter = 1000,
  details = 0,
  ...
)
```

Arguments

`S`	Empirical covariance matrix
`n.obs`	Number of observations
`glist`	Generating class for model (a list)
`start`	Initial value for concentration matrix
`eps`	Convergence criterion
`iter`	Maximum number of iterations
`details`	Controlling the amount of output.
`...`	Optional arguments; currently not used

Details

`ggmfit` is based on a C implementation. `ggmfitr` is implemented purely in R (and is provided mainly as a benchmark for the C-version).

Value

A list with

`lrt`	Likelihood ratio statistic (-2logL)
`df`	Degrees of freedom
`logL`	log likelihood
`K`	Estimated concentration matrix (inverse covariance matrix)

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

`cmod`, `loglin`

Examples

```
## Fitting "butterfly model" to mathmark data
## Notice that the output from the two fitting functions is not
## entirely identical.
data(math)
ddd <- cov.wt(math, method="ML")
glist <- list(c("al", "st", "an"), c("me", "ve", "al"))
ggmfit (ddd$cov, ddd$n.obs, glist)
ggmfitr(ddd$cov, ddd$n.obs, glist)
```

`imodel-dmod` *Discrete interaction model (log-linear model)*

Description

Specification of log–linear (graphical) model. The 'd' in the name `dmod` refers to that it is a (graphical) model for 'd'iscrete variables

Usage

```
dmod(
  formula,
  data,
  marginal = NULL,
  interactions = NULL,
  fit = TRUE,
  details = 0,
  ...
)
```

Arguments

`formula`	Model specification in one of the following forms: 1) a right-hand sided formula, 2) as a list of generators, 3) an undirected graph (represented either as a graphNEL object or as an adjacency matrix). Notice that there are certain model specification shortcuts, see Section 'details' below.
`data`	Either a table or a dataframe. In the latter case, the dataframe will be coerced to a table. See 'details' below.

`marginal`	Should only a subset of the variables be used in connection with the model specification shortcuts
`interactions`	A number given the highest order interactions in the model, see Section 'details' below.
`fit`	Should the model be fitted.
`details`	Control the amount of output; for debugging purposes.
`...`	Additional arguments; currently no used.

Details

The independence model can be specified as `~.^1` and `~.^.` specifies the saturated model. Setting e.g. `interactions=3` implies that there will be at most three factor interactions in the model.

Data can be specified as a table of counts or as a dataframe. If data is a dataframe then it will be converted to a table (using `xtabs()`). This means that if the dataframe contains numeric values then the you can get a very sparse and high dimensional table. When a dataframe contains numeric values it may be worthwhile to discretize data using the `cut()` function.

The `marginal` argument can be used for specifying the independence or saturated models for only a subset of the variables. When `marginal` is given the corresponding marginal table of data is formed and used in the analysis (notice that this is different from the behaviour of `loglin()` which uses the full table.

The `triangulate()` method for discrete models (dModel objects) will for a model look at the dependence graph for the model.

Value

An object of class `dModel`.

Author(s)

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

See Also

`cmod`, `mmod`

Examples

```
## Graphical log-linear model
data(reinis)
dm1 <- dmod(~ .^., reinis)
dm2 <- backward(dm1, k=2)
dm3 <- backward(dm1, k=2, fixin=list(c("family", "phys", "systol")))
## At most 3-factor interactions
dm1<-dmod(~ .^., data=reinis, interactions=3)
```

`imodel-general` *General functions related to iModels*

Description

General functions related to iModels

Usage

```
## S3 method for class 'iModel'
logLik(object, ...)

## S3 method for class 'iModel'
extractAIC(fit, scale, k = 2, ...)

## S3 method for class 'iModel'
summary(object, ...)

## S3 method for class 'iModelsummary'
print(x, ...)

## S3 method for class 'iModel'
formula(x, ...)

## S3 method for class 'iModel'
terms(x, ...)

## S3 method for class 'dModel'
isGraphical(x)

## S3 method for class 'dModel'
isDecomposable(x)

modelProperties(object)

## S3 method for class 'dModel'
modelProperties(object)
```

Arguments

`object, fit, x`	An `iModel` object.
`...`	Currently unused.
`scale`	Unused (and irrelevant for these models)
`k`	Weight of the degrees of freedom in the AIC formula

`imodel-info` *Get information about mixed interaction model objects*

Description

General functions related to iModels

Usage

```
getmi(object, name)
```

Arguments

`object`	An `iModel` object.
`name`	The slot / information to be extracted.

`imodel-mmod` *Mixed interaction model.*

Description

A mixed interaction model is a model (often with conditional independence restrictions) for a combination of discrete and continuous variables.

Usage

```
mmod(formula, data, marginal = NULL, fit = TRUE, details = 0)
```

Arguments

`formula`	A right hand sided formula specifying the model.
`data`	Data (a dataframe)
`marginal`	A possible subsets of columns of `data`; useful when `formula` contains model specification shortcuts.
`fit`	Currently not used
`details`	For printing debugging information

Value

An object of class `mModel` and the more general class `iModel`.

Author(s)

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

See Also

`dmod`, `cmod`.

Examples

```
### FIXME: To be written
```

`internal` *Internal functions for the gRim package*

Description

Internal functions for the gRim package

`loglin-dim` *Return the dimension of a log-linear model*

Description

Return the dimension of a log-linear model given by the generating class 'glist'. If the model is decomposable and adjusted dimension can be found.

Usage

```
dim_loglin(glist, tableinfo)

dim_loglin_decomp(glist, tableinfo, adjust = TRUE)
```

Arguments

`glist`	Generating class (a list) for a log-linear model. See 'details' below.
`tableinfo`	Specification of the levels of the variables. See 'details' below.
`adjust`	Should model dimension be adjusted for sparsity of data (only available for decomposable models)

Details

`glist` can be either a list of vectors with variable names or a list of vectors of variable indices.

`tableinfo` can be one of three different things.

1. A contingency table (a `table`).
2. A list with the names of the variables and their levels (such as one would get if calling `dimnames` on a `table`).

3. A vector with the levels. If `glist` is a list of vectors with variable names, then the entries of the vector `tableinfo` must be named.

If the model is decomposable it `dim_loglin_decomp` is to be preferred over `dim_loglin` as the former is much faster.

Setting `adjust=TRUE` will force `dim_loglin_decomp` to calculated a dimension which is adjusted for sparsity of data. For this to work, `tableinfo` *MUST* be a table.

Value

A numeric.

Author(s)

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

See Also

`dmod`, `glm`, `loglm`

Examples

```
## glist contains variable names and tableinfo is a named vector:
dim_loglin(list(c("a", "b"), c("b", "c")), c(a=4, b=7, c=6))

## glist contains variable names and tableinfo is not named:
dim_loglin(list(c(1, 2), c(2, 3)), c(4, 7, 6))

## For decomposable models:
dim_loglin_decomp(list(c("a", "b"), c("b", "c")), c(a=4, b=7, c=6),adjust=FALSE)
```

`loglin-effloglin` *Fitting Log-Linear Models by Message Passing*

Description

Fit log-linear models to multidimensional contingency tables by Iterative Proportional Fitting.

Usage

```
effloglin(table, margin, fit = FALSE, eps = 0.01, iter = 20, print = TRUE)
```

Arguments

`table`	A contingency table
`margin`	A generating class for a hierarchical log–linear model
`fit`	If TRUE, the fitted values are returned.
`eps`	Convergence limit; see 'details' below.
`iter`	Maximum number of iterations allowed
`print`	If TRUE, iteration details are printed.

Details

The function differs from `loglin` in that 1) data can be given in the form of a list of sufficient marginals and 2) the model is fitted only on the cliques of the triangulated interaction graph of the model. This means that the full table is not fitted, which means that `effloglin` is efficient (in terms of storage requirements). However `effloglin` is implemented entirely in R and is therefore slower than `loglin`. Argument names are chosen so as to match those of loglin()

Value

A list.

Author(s)

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

References

Radim Jirousek and Stanislav Preucil (1995). On the effective implementation of the iterative proportional fitting procedure. Computational Statistics & Data Analysis Volume 19, Issue 2, February 1995, Pages 177-189

See Also

`loglin`

Examples

```
data(reinis)
glist <-list(c("smoke", "mental"), c("mental", "phys"),
             c("phys", "systol"), c("systol", "smoke"))

stab <- lapply(glist, function(gg) tabMarg(reinis, gg))
fv3 <- effloglin(stab, glist, print=FALSE)
```

`modify_glist` *Modify generating class for a graphical/hierarchical model*

Description

Modify generating class for a graphical/hierarchical model by 1) adding edges, 2) deleting edges, 3) adding terms and 4) deleting terms.

Usage

```
modify_glist(glist, items, details = 0)
```

Arguments

`glist`	A list of vectors where each vector is a generator of the model.
`items`	A list with edges / terms to be added and deleted. See section 'details' below.
`details`	Control the amount of output (for debugging purposes).

Details

The `items` is a list with named entries as `list(add.edge=,drop.edge=, add.term=, drop.term=)`

Not all entries need to be in the list. The corresponding actions are carried out in the order in which they appear in the list.

See section 'examples' below for examples.

Notice that the operations do not in general commute: Adding an edge which is already in a generating class and then removing the edge again does not give the original generating class.

Value

A generating class for the modified model. The elements of the list are character vectors.

Author(s)

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

See Also

`cmod`, `dmod`, `mmod`

Examples

```
glist <- list(c(1, 2, 3), c(2, 3, 4))

## Add edges
modify_glist(glist, items=list(add.edge=c(1, 4)))
modify_glist(glist, items=list(add.edge=~1:4))
```

```
## Add terms
modify_glist(glist, items=list(add.term=c(1, 4)))
modify_glist(glist, items=list(add.term=~1:4))

## Notice: Only the first term is added as the second is already
## in the model.
modify_glist(glist, items=list(add.term=list(c(1, 4), c(1, 3))))
modify_glist(glist, items=list(add.term=~1:4 + 1:3))

## Notice: Operations are carried out in the order given in the
## items list and hence we get different results:
modify_glist(glist, items=list(drop.edge=c(1, 4), add.edge=c(1, 4)))
modify_glist(glist, items=list(add.edge=c(1, 4), drop.edge=c(1, 4)))
```

`parm-conversion` *Conversion between different parametrizations of mixed models*

Description

Functions to convert between canonical parametrization (g,h,K), moment parametrization (p,m,S) and mixed parametrization (p,h,K).

Usage

```
parm_pms2ghk(parms)

parm_ghk2pms(parms)

parm_pms2phk(parms)

parm_phk2ghk(parms)

parm_phk2pms(parms)

parm_ghk2phk(parms)

parm_CGstats2mmod(parms, type = "ghk")

parm_moment2pms(SS)
```

Arguments

`parms`	Parameters of a mixed interaction model
`type`	Output parameter type; either "ghk" or "pms".
`SS`	List of moment parameters.

Value

Parameters of a mixed interaction model.

Author(s)

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

`parse_gm_formula` *Parse graphical model formula*

Description

Parse graphical model formula to internal representation

Usage

```
parse_gm_formula(
  formula,
  varnames = NULL,
  marginal = NULL,
  interactions = NULL
)
```

Arguments

`formula`	A right hand sided formula or a list.
`varnames`	Specification of the variables.
`marginal`	Possible specification of marginal (a set of variables); useful in connection with model specification shortcuts.
`interactions`	The maximum order of interactions allowed; useful in connection with model specification shortcuts.

Examples

```
vn <- c("me", "ve", "al", "an", "st")

form1 <- ~me:ve:al + ve:al + an
form2 <- ~me:ve:al + ve:al + s
form3 <- ~me:ve:al + ve:al + anaba
parse_gm_formula(form1, varnames=vn)
parse_gm_formula(form2, varnames=vn)
## parse_gm_formula(form3, varnames=vn)
parse_gm_formula(form1)
parse_gm_formula(form2)
parse_gm_formula(form3)

## parse_gm_formula(~.^1)
```

```
## parse_gm_formula(~.^.)

parse_gm_formula(~.^1, varnames=vn)
parse_gm_formula(~.^., varnames=vn)
parse_gm_formula(~.^., varnames=vn, interactions=3)

vn2 <- vn[1:3]
## parse_gm_formula(form1, varnames=vn, marginal=vn2)
## parse_gm_formula(form2, varnames=vn, marginal=vn2)
## parse_gm_formula(form3, varnames=vn, marginal=vn2)
parse_gm_formula(~.^1, varnames=vn, marginal=vn2)
parse_gm_formula(~.^., varnames=vn, marginal=vn2)
```

`stepwise` *Stepwise model selection in (graphical) interaction models*

Description

Stepwise model selection in (graphical) interaction models

Usage

```
drop_func(criterion)

## S3 method for class 'iModel'
stepwise(
  object,
  criterion = "aic",
  alpha = NULL,
  type = "decomposable",
  search = "all",
  steps = 1000,
  k = 2,
  direction = "backward",
  fixin = NULL,
  fixout = NULL,
  details = 0,
  trace = 2,
  ...
)

backward(
  object,
  criterion = "aic",
  alpha = NULL,
  type = "decomposable",
  search = "all",
```

```
  steps = 1000,
  k = 2,
  fixin = NULL,
  details = 1,
  trace = 2,
  ...
)

forward(
  object,
  criterion = "aic",
  alpha = NULL,
  type = "decomposable",
  search = "all",
  steps = 1000,
  k = 2,
  fixout = NULL,
  details = 1,
  trace = 2,
  ...
)
```

Arguments

`criterion`	Either `"aic"` or `"test"` (for significance test)
`object`	An `iModel` model object
`alpha`	Critical value for deeming an edge to be significant/ insignificant. When `criterion="aic"`, `alpha` defaults to 0; when `criterion="test"`, `alpha` defaults to 0.05.
`type`	Type of models to search. Either `"decomposable"` or `"unrestricted"`. If `type="decomposable"` and the initial model is decompsable, then the search is among decomposable models only.
`search`	Either `'all'` (greedy) or `'headlong'` (search edges randomly; stop when an improvement has been found).
`steps`	Maximum number of steps.
`k`	Penalty term when `criterion="aic"`. Only k=2 gives genuine AIC.
`direction`	Direction for model search. Either `"backward"` or `"forward"`.
`fixin`	Matrix (p x 2) of edges. If those edges are in the model, they are not considered for removal.
`fixout`	Matrix (p x 2) of edges. If those edges are not in the model, they are not considered for addition.
`details`	Controls the level of printing on the screen.
`trace`	For debugging only
`...`	Further arguments to be passed on to `testdelete` (for `testInEdges`) and `testadd` (for `testOutEdges`).

Author(s)

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

See Also

`cmod`, `dmod`, `mmod`, `testInEdges`, `testOutEdges`

Examples

```
data(reinis)
## The saturated model
m1 <- dmod(~.^., data=reinis)
m2 <- stepwise(m1)
m2
```

`test-edges` *Test edges in graphical models with p-value/AIC value*

Description

Test edges in graphical models with p-value/AIC value. The models must be `iModels`.

Usage

```
testEdges(
  object,
  edgeMAT = NULL,
  ingraph = TRUE,
  criterion = "aic",
  k = 2,
  alpha = NULL,
  headlong = FALSE,
  details = 1,
  ...
)

testInEdges(
  object,
  edgeMAT = NULL,
  criterion = "aic",
  k = 2,
  alpha = NULL,
  headlong = FALSE,
  details = 1,
  ...
)
```

```
testOutEdges(
  object,
  edgeMAT = NULL,
  criterion = "aic",
  k = 2,
  alpha = NULL,
  headlong = FALSE,
  details = 1,
  ...
)
```

Arguments

`object`	An `iModel` model object
`edgeMAT`	A `p * 2` matrix with edges
`ingraph`	If TRUE, edges in graph are tested; if FALSE, edges not in graph are tested.
`criterion`	Either `"aic"` or `"test"` (for significance test)
`k`	Penalty term when `criterion="aic"`. Only k=2 gives genuine AIC.
`alpha`	Critical value for deeming an edge to be significant/ insignificant. When `criterion="aic"`, `alpha` defaults to 0; when `criterion="test"`, `alpha` defaults to 0.05.
`headlong`	If TRUE then testing will stop once a model improvement has been found.
`details`	Controls the level of printing on the screen.
`...`	Further arguments to be passed on to `testdelete` (for `testInEdges`) and `testadd` (for `testOutEdges`).

Details

- testIn: Function which tests whether each edge in "edgeList" can be delete from model "object"
- testOut: Is similar but in the other direction.

Value

A dataframe with test statistics (p-value or change in AIC), edges and logical telling if the edge can be deleted.

Author(s)

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

See Also

`getEdges`, `testadd`, `testdelete`

Examples

```
data(math)
cm1 <- cmod(~me:ve + ve:al + al:an, data=math)
testEdges(cm1, ingraph=TRUE)
testEdges(cm1, ingraph=FALSE)
## Same as
# testInEdges(cm1)
# testOutEdges(cm)
```

`testadd` *Test addition of edge to graphical model*

Description

Performs a test of addition of an edge to a graphical model (an `iModel` object).

Usage

```
testadd(object, edge, k = 2, details = 1, ...)
```

Arguments

`object`	A model; an object of class `iModel`.
`edge`	An edge; either as a vector or as a right hand sided formula.
`k`	Penalty parameter used when calculating change in AIC
`details`	The amount of details to be printed; 0 surpresses all information
`...`	Further arguments to be passed on to the underlying functions for testing.

Details

Let M0 be the model and e=u,v be an edge and let M1 be the model obtained by adding e to M0. If M1 is decomposable AND e is contained in one clique C only of M1 then the test is carried out in the C-marginal model. In this case, and if the model is a log-linear model then the degrees of freedom is adjusted for sparsity.

Value

A list

Author(s)

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

See Also

`testdelete`

Examples

```
## Discrete models
data(reinis)

## A decomposable model
mf <- ~smoke:phys:mental + smoke:systol:mental
object <- dmod(mf, data=reinis)
testadd(object, c("systol", "phys"))

## A non-decomposable model
mf <- ~smoke:phys + phys:mental + smoke:systol + systol:mental
object <- dmod(mf, data=reinis)
testadd(object, c("phys", "systol"))

## Continuous models
data(math)

## A decomposable model
mf <- ~me:ve:al + al:an
object <- cmod(mf, data=math)
testadd(object, c("me", "an"))

## A non-decomposable model
mf <- ~me:ve + ve:al + al:an + an:me
object <- cmod(mf, data=math)
testadd(object, c("me", "al"))
```

`testdelete` *Test deletion of edge from an interaction model*

Description

Tests if an edge can be deleted from an interaction model.

Usage

```
testdelete(object, edge, k = 2, details = 1, ...)
```

Arguments

`object`	A model; an object of class `iModel`.
`edge`	An edge in the model; either as a right-hand sided formula or as a vector
`k`	Penalty parameter used when calculating change in AIC
`details`	The amount of details to be printed; 0 surpresses all information
`...`	Further arguments to be passed on to the underlying functions for testing.

Details

If the model is decomposable and the edge is contained in one clique only then the test is made in the marginal model given by that clique. In that case, if the model is a log-linear model then degrees of freedom are adjusted for sparsity

If model is decomposable and edge is in one clique only, then degrees of freedom are adjusted for sparsity

Value

A list.

Author(s)

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

See Also

`testadd`

Examples

```
## Discrete models
data(reinis)

## A decomposable model
mf <- ~smoke:phys:mental + smoke:systol:mental
object <- dmod(mf, data=reinis)
testdelete(object, c("phys", "mental"))
testdelete(object, c("smoke", "mental"))

## A non-decomposable model
mf <- ~smoke:phys + phys:mental + smoke:systol + systol:mental
object <- dmod(mf, data=reinis)

testdelete(object, c("phys", "mental"))

## Continuous models
data(math)

## A decomposable model
mf <- ~me:ve:al + me:al:an
object <- cmod(mf, data=math)
testdelete(object, c("ve", "al"))
testdelete(object, c("me", "al"))

## A non-decomposable model
mf <- ~me:ve + ve:al + al:an + an:me
object <- cmod(mf, data=math)
testdelete(object, c("me", "ve"))
```

Index