Package ‘gRim’

July 19, 2024

Version 0.3.3
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)
Depends R (>= 4.2.0), methods, gRain (>= 1.3.10), gRbase (>= 2.0.2)
Suggests testthat (>= 2.1.0), markdown, knitr
Imports doBy, igraph, stats4, glue, Matrix, MASS, Rcpp (>= 0.11.1)
URL https://people.math.aau.dk/~sorenh/software/gR/
Encoding UTF-8
VignetteBuilder knitr
ByteCompile yes
LinkingTo Rcpp (>= 0.11.1), RcppArmadillo, RcppEigen, gRbase (>= 2.0.2)
RoxygenNote 7.3.1
NeedsCompilation yes
Repository CRAN
Date/Publication 2024-07-19 14:40:02 UTC

Contents

cg-stats ................................................................. 2
citest-array ............................................................ 3
citest-df ................................................................. 5
**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)
Test for conditional independence in a contingency table

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 conditionally 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 Pear-
    sons 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>
citest-df

Test for conditional independence in a dataframe

Description

Test for conditional independence in a dataframe.

Usage

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

See Also

ciTest, ciTest_df, ciTest_mvn, chisq.test
Arguments

x
A dataframe.

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

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)
ciTTest_df(milkcomp1, set=~tre + fat + pro)
Generic function for conditional independence test. Specializes to specific types of data.

Usage

\texttt{ciTest(x, set = NULL, ...)}

Arguments

- \texttt{x}: An object for which a test for conditional independence is to be made. See 'details' for valid types of \texttt{x}.
- \texttt{set}: A specification of the test to be made. The tests are of the form \(u\) and \(v\) are independent conditionally on \(S\) where \(u\) and \(v\) are variables and \(S\) is a set of variables. See 'details' for details about specification of \texttt{set}.
- \texttt{...}: Additional arguments to be passed on to other methods.

Details

\texttt{x} can be

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

\texttt{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 \texttt{citest} (which is a list).

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

\texttt{ciTest_table, ciTest_df, ciTest_mvn, chisq.test}
Examples

```r
## 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

```r
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 conditionally 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.
citest-ordinal

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 Also

ciTest_table, ciTest

Examples

```r
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)
```
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

```r
cmod(
  formula,
  data,
  marginal = NULL,
  fit = TRUE,
  maximal_only = FALSE,
  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.
- **maximal_only**: Should only maximal generators be retained.
- **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

```r
## 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)),
                              "LeanMeat", "Fat11", "Fat12", "Fat13"),
                              ncol=2))
```

Description

Coerce models to different representations

Usage

```r
as_emat2cq(emat, nvar = NULL)
as_emat_complement(emat, nvar)
as_emat2amat(emat, d)
as_emat2elist(emat)
as_elist2emat(elist)
as_glist2emat(glist)
as_glist2emat(glist)
as_glist2graph(glist, d)
as_glist2igraph(glist, d)
as_emat2graph(emat, d)
```
as_emat2igraph(emat, d)

as_amat2emat(amat, eps = 1e-04)

as_emat2glist(emat)

as_glist2out_edges(glist)

as_K2amat(K, eps = 1e-04)

as_K2graph(K)

as_sparse(K)

**Arguments**

- **emat**: Edge matrix (2 x p)
- **nvar**: Number of variables
- **d**: Number of columns in output.
- **elist**: Edge list (list of pairs)
- **glist**: Generator list
- **amat**: Adjacency matrix
- **eps**: Small number

```r
Glist <- list(c(1,2,3),c(2,3,4),c(5,6)) em <- as_glist2emat(glist) am <- as_emat2amat(em, d=6) ig <- as_emat2igraph(em) el <- as_emat2elist(em) igraph::max_cliques(ig) as_emat2cq(em, 6) as_emat_complement(em, 6)
```

- **K**: Concentration matrix

**Description**

Edge matrix operations needed for ips algorithms

**Usage**

- `emat_compare(emat1, emat2)`
- `emat_complement(emat1, emat2)`
- `emat_sort(emat1)`
- `order_rows(emat)`
fast_cov

Arguments
emat, emat1, emat2

Edge matrix (a 2 x p matrix)

Details
An emat with p edges is represented by a 2 x p matrix.

Note
These functions may well be removed from the package in future releases

Author(s)
Søren Højsgaard, <sorenh@math.aau.dk>

Examples
emat1 <- model_saturated(3:4, type="emat")
emat2 <- model_saturated(1:4, type="emat")
emat_complement(emat1, emat2)
emat3 <- model_saturated(2:4, type="emat")
emat_compare(emat1, emat3)

---

fast_covFast computation of covariance / correlation matrix

Description
Fast computation of covariance / correlation matrix

Usage
fast_cov(x, center = TRUE, scale = TRUE)

Arguments
x a numeric matrix(like object).
center, scale Should columns in x be centered and/or scaled
Description

Fit Gaussian graphical models using various algorithms.

Usage

```r
fit_ggm_grips(
  S,
  formula = NULL,
  nobs,
  K = NULL,
  maxit = 10000L,
  eps = 0.01,
  convcrit = 1,
  aux = list(),
  method = "ncd",
  print = 0
)
```

Arguments

- **S**: Sample covariance matrix.
- **formula**: Generators of model; a list of integer vectors or a 2 x p matrix of integers.
- **nobs**: Number of observations.
- **K**: Initial value of concentration matrix.
- **maxit**: Maximum number of iterations.
- **eps**: Convergence criterion.
- **convcrit**: Convergence criterions. See section details.
- **aux**: A list of form name=value.
- **method**: Either "ncd" (default), "covips" or "conips".
- **print**: Should output from fitting be printed?

Details

Convergence criterion:

- 1: max absolute difference between S and SigmaHat on edges.
- 2: difference in log likelihood divided by number of parameters in the model (number of edges + number of nodes) between successive iterations.
- 3: computed duality gap may turn negative due to rounding error, so its absolute value is returned. This still provides upper bound on error of likelihood function.
Methods:

- "ncd": Neighbour coordinate descent.
- "covips": IPS based on working with the covariance matrix.
- "conips": IPS based on working with the concentration matrix.

ncd is very fast but may fail to converge in rare cases. Both covips and conips are guaranteed to converge provided the maximum likelihood estimate exists, and covips are considerably faster than conips.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```r
options("digits"=3)
data(math, package="gRbase")
S <- cov(math)
nobs <- nrow(math)
g1 <- list(1:3, 3:5)
em <- matrix(c(1,2, 2,3, 1,3, 3,4, 3,5, 4,5), nrow=2)
EPS = 1e-2

fit_cov = fit_ggm_grips(S, g1, nobs=nobs, eps=EPS, method="cov")
fit_con = fit_ggm_grips(S, g1, nobs=nobs, eps=EPS, method="con")
fit_ncd = fit_ggm_grips(S, g1, nobs=nobs, eps=EPS, method="ncd")

K <- solve(S)
(fit_con$K - K) |> abs() |> max()
(fit_cov$K - K) |> abs() |> max()
(fit_ncd$K - K) |> abs() |> max()
```

---

generate_models  Generate various graphical models

Description

Models are represented in various forms

Usage

emat_saturated_model(index)

model_saturated(index, type = "emat", nms = NULL)
model_random_tree(index, prob = 0, type = "emat", nms = NULL)
model_rectangular_grid(dim, type = "emat", nms = NULL)
model_line(index, type = "emat", nms = NULL)
model_star(index, type = "emat", nms = NULL)
model_loop(index, prob = 0, type = "emat", nms = NULL)
model_random(index, prob = 0.1, type = "emat", nms = NULL)

Arguments
index A vector of integers
type Output type.
nms Names of variables.
prob Probability of any edge being present.
dim A vector with dimensions

generate_n01  Genrate matrix of N(0, 1) variables

Description
Genrate matrix of N(0, 1) variables

Usage
generate_n01(n.obs, nvar, seed = 2022)

Arguments
n.obs Number of rows
nvar Number of columns
seed Seed for random number generator
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 an igraph 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, an igraph 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>
getEdges

See Also

edgeList, nonEdgeList.

Examples

```r
gg <- ug(~a:b:d + a:c:d + c:e, result="igraph")
glist <- get Cliques(gg)
adjmat <- as(gg, "matrix")

#### 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, result="igraph")
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**

```r
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**

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

**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**

```r
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 an igraph 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
dml1<-dmod(~ .", data=reinis, interactions=3)
Description

General functions related to iModels

Usage

```r
## 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 'iModel'
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>
impose_zero

See Also
dmod, cmod.

Examples

### FIXME: To be written

---

**impose_zero**  
*Impose zeros in matrix entries which do not correspond to an edge.*

---

**Description**

Impose zeros in matrix entries which do not correspond to an edge.

**Usage**

```r
impose_zero(emat, K)
```

**Arguments**

- `emat`: Edge matrix (2 x p matrix)
- `K`: Matrix; typically a concentration matrix.

---

**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>
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

See Also
loglin

Examples
```r
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)))
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,
  maximal_only = FALSE
)

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.
- **maximal_only**: Should only maximal generators be retained.

Examples

```r
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)
pase_gm_formula(~.^1, varnames=vn, marginal=vn2)
pase_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,
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 decomposable, 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

cmd, 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.

Details

• testIn: Function which tests whether each edge in "edgeList" can be deleted 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

\[
mf <- \text{smoke:phys + phys:mental + smoke:systol + systol:mental}
\]

\[
object <- \text{dmod(mf, data=reinis)}
\]

\[
testadd(object, c(\text{"phys", \"systol\")})
\]

## Continuous models

data(math)

## A decomposable model

\[
mf <- \text{me:ve:al + al:an}
\]

\[
object <- \text{cmod(mf, data=math)}
\]

\[
testadd(object, c(\text{\"me\", \"an\")})
\]

## A non-decomposable model

\[
mf <- \text{me:ve + ve:al + al:an + an:me}
\]

\[
object <- \text{cmod(mf, data=math)}
\]

\[
testadd(object, c(\text{\"me\", \"al\")})
\]

---

### testdelete

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

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

**Usage**

\[
testdelete(object, edge, k = 2, details = 1, \ldots)
\]

**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 suppresses all information
- **\ldots**: 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 the 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

```r
## 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"))
```
Usage

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

## S3 method for class 'gips_fit_class'
AIC(object, ..., k = 2)

## S3 method for class 'gips_fit_class'
BIC(object, ...)

## S3 method for class 'gips_fit_class'
sigma(object, ...)

concentration(object, ...)

## S3 method for class 'gips_fit_class'
concentration(object, ...)

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

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

glance.gips_fit_class(x, ...)

Arguments

object      Model object.
...         Additional arguments; currently not used.
k           Penalty parameter for calculating AIC; only k=2 gives genuine AIC.
x           Object to be printed.
Index

* htest
  citest-array, 3
citest-df, 5
citest-generic, 7
citest-mvn, 8
citest-ordinal, 9
test-edges, 34
testadd, 36
testdelete, 37
* models
  cmod, 11
ggmfit, 20
  imodel-dmod, 21
  imodel-mmod, 24
loglin-dim, 26
loglin-effloglin, 27
stepwise, 32
test-edges, 34
  testadd, 36
testdelete, 37
* multivariate
  ggmfit, 20
* utilities
  cg-stats, 2
  CGstats (cg-stats), 2
chisq.test, 5–7, 9
ciTest, 5, 6, 9, 10
ciTest (citest-generic), 7
citest-array, 3
citest-df, 5
citest-generic, 7
citest-mvn, 8
citest-ordinal, 9
ciTest_df, 5, 7, 9
ciTest_df (citest-df), 5
ciTest_mvn, 5–7, 9
ciTest_mvn (citest-mvn), 8
ciTest_ordinal (citest-ordinal), 9
ciTest_table, 6, 7, 9, 10
ciTest_table (citest-array), 3
  cmod, 11, 21, 22, 25, 29, 33
  coef.mModel (imodel-mmod), 24
  coefficients.mModel (imodel-mmod), 24
coeerce_models, 12
concentration (utilities_grips), 38
cov.wt, 3
dim_loglin (loglin-dim), 26
dim_loglin_decomp (loglin-dim), 26
dmod, 12, 25, 27, 29, 33
dmod (imodel-dmod), 21
drop_func (stepwise), 32
AIC.gips_fit_class (utilities_grips), 38
  as_glist2emat (coerce_models), 12
  as_glist2graph (coerce_models), 12
  as_glist2i amat (coerce_models), 12
  as_glist2out_edges (coerce_models), 12
  as_K2amat (coerce_models), 12
  as_K2graph (coerce_models), 12
  as_sparse (coerce_models), 12
backward (stepwise), 32
BIC.gips_fit_class (utilities_grips), 38
  cg-stats, 2
  CGstats (cg-stats), 2
  chisq.test, 5–7, 9
  ciTest, 5, 6, 9, 10
  ciTest (citest-generic), 7
  citest-array, 3
  citest-df, 5
  citest-generic, 7
  citest-mvn, 8
  citest-ordinal, 9
  ciTest_df, 5, 7, 9
  ciTest_df (citest-df), 5
  ciTest_mvn, 5–7, 9
  ciTest_mvn (citest-mvn), 8
  ciTest_ordinal (citest-ordinal), 9
  ciTest_table, 6, 7, 9, 10
  ciTest_table (citest-array), 3
  cmod, 11, 21, 22, 25, 29, 33
  coef.mModel (imodel-mmod), 24
  coefficients.mModel (imodel-mmod), 24
coeerce_models, 12
concentration (utilities_grips), 38
cov.wt, 3
dim_loglin (loglin-dim), 26
dim_loglin_decomp (loglin-dim), 26
dmod, 12, 25, 27, 29, 33
dmod (imodel-dmod), 21
drop_func (stepwise), 32
AIC.gips_fit_class (utilities_grips), 38
  as_glist2emat (coerce_models), 12
  as_glist2graph (coerce_models), 12
  as_glist2i amat (coerce_models), 12
  as_glist2out_edges (coerce_models), 12
  as_K2amat (coerce_models), 12
  as_K2graph (coerce_models), 12
  as_sparse (coerce_models), 12
backward (stepwise), 32
BIC.gips_fit_class (utilities_grips), 38
  cg-stats, 2
  CGstats (cg-stats), 2
  chisq.test, 5–7, 9
  ciTest, 5, 6, 9, 10
  ciTest (citest-generic), 7
  citest-array, 3
  citest-df, 5
  citest-generic, 7
  citest-mvn, 8
  citest-ordinal, 9
  ciTest_df, 5, 7, 9
  ciTest_df (citest-df), 5
  ciTest_mvn, 5–7, 9
  ciTest_mvn (citest-mvn), 8
  ciTest_ordinal (citest-ordinal), 9
  ciTest_table, 6, 7, 9, 10
  ciTest_table (citest-array), 3
  cmod, 11, 21, 22, 25, 29, 33
  coef.mModel (imodel-mmod), 24
  coefficients.mModel (imodel-mmod), 24
coeerce_models, 12
concentration (utilities_grips), 38
cov.wt, 3
dim_loglin (loglin-dim), 26
dim_loglin_decomp (loglin-dim), 26
dmod, 12, 25, 27, 29, 33
dmod (imodel-dmod), 21
drop_func (stepwise), 32
INDEX

edgeList, 19
effloglin (loglin-effloglin), 27
emat_compare (emat_operations), 13
emat_complement (emat_operations), 13
emat_operations, 13
emat_saturated_model (generate_models), 16
emat_sort (emat_operations), 13
extract_cmod_data (cmod), 11
extractAIC.iModel (imodel-general), 23
fast_cov, 14
fit_ggm_grips, 15
fitted.dModel (imodel-dmod), 21
formula.iModel (imodel-general), 23
forward (stepwise), 32
generate_models, 16
generate_n01, 17
getEdges, 18, 35
getEdgesMAT (getEdges), 18
getInEdges (getEdges), 18
getInEdgesMAT (getEdges), 18
getmi (imodel-info), 24
getOutEdges (getEdges), 18
getOutEdgesMAT (getEdges), 18
ggmfit, 12, 20
ggmfitr (ggmfit), 20
glance.gips_fit_class (utilities_grips), 38
glm, 27
imodel-dmod, 21
imodel-general, 23
imodel-info, 24
imodel-mmod, 24
impose_zero, 25
internal, 25
isDecomposabe.dModel (imodel-general), 23
isGraphical.dModel (imodel-general), 23
logLik.gips_fit_class (utilities_grips), 38
logLik.iModel (imodel-general), 23
loglin, 27, 28
loglin-dim, 26
loglin-effloglin, 27
loglm, 27
mcs_marked, 18
mmod, 12, 22, 29, 33
mmod (imodel-mmod), 24
mmod_dimension (imodel-mmod), 24
model_line (generate_models), 16
model_loop (generate_models), 16
model_random (generate_models), 16
model_random_tree (generate_models), 16
model_rectangular_grid (generate_models), 16
model_saturated (generate_models), 16
model_star (generate_models), 16
modelProperties (imodel-general), 23
modify_glist, 28
nonEdgeList, 19
order_rows (emat_operations), 13
parm-conversion, 30
parm_CGstats2mmod (parm-conversion), 30
parm_ghk2phk (parm-conversion), 30
parm_ghk2pms (parm-conversion), 30
parm_moment2pms (parm-conversion), 30
parm_phk2ghk (parm-conversion), 30
parm_phk2pms (parm-conversion), 30
parm_pms2ghk (parm-conversion), 30
parm_pms2pms (parm-conversion), 30
parse_gm_formula, 31
print.dModel (imodel-dmod), 21
print.gips_fit_class (utilities_grips), 38
print.iModel (imodel-general), 23
print.mModel (imodel-mmod), 24
print.testadd (testadd), 36
print.testdelete (testdelete), 37
residuals.dModel (imodel-dmod), 21
sigma.gips_fit_class (utilities_grips), 38
stepwise, 32
summary.gips_fit_class (utilities_grips), 38
summary.iModel (imodel-general), 23
summary.mModel (imodel-mmod), 24
terms.iModel (imodel-general), 23
test-edges, 34
testadd, 35, 36, 38
testdelete, 35, 36, 37

testEdges (test-edges), 34

testInEdges, 33

testInEdges (test-edges), 34

testOutEdges, 33

testOutEdges (test-edges), 34

utilities_grips, 38