Package 'algstat'

January 28, 2019

Type Package

Title Algebraic statistics in R

Version 0.0.2

Date 2014-12-04

Maintainer David Kahle <david.kahle@gmail.com>

Description algstat provides functionality for algebraic statistics in R. Current applications include exact inference in log-linear models for contingency table data, analysis of ranked and partially ranked data, and general purpose tools for multivariate polynomials, building on the mpoly package. To aid in the process, algstat has ports to Macaulay2, Bertini, LattE-integrale and 4ti2.

Depends mpoly

LinkingTo Rcpp

Imports stringr, reshape2, Rcpp

License GPL-2

SystemRequirements Optionally Latte-integrale, Bertini, and Macaulay2. Cygwin is required for each of the above for Windows users. See INSTALL file for details.

Author David Kahle [aut, cre], Luis Garcia-Puente [aut]

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-12-06 02:23:38

R topics documented:

abortion — *Haberman's Abortion, Education, and Religion Data*

Description

A multi-way contingency table with the results of a survey concerning attitudes concerning abortion.

Usage

```
data(abortion)
```

Format

A 3x3x3 (contingency) table

Author(s)

1972 National Opinion Research Center

References

Haberman, S. (1978). *Analysis of Qualitative Data* 1, 2. Academic Press, Orlando, FL.

algstat — *algstat : Algebraic statistics in R*

Description

algstat is a package for algebraic statistics in R. Current applications include exact inference in log-linear models for contingency table data, analysis of ranked and partially ranked data, and general purpose tools for multivariate polynomials, building on the mpoly package. To aid in the process, algstat has ports to Macaulay2, Bertini, Latte-Integrale and 4ti2.

Amaker — *Distance transitive matrix*

Description

Compute the distance transitive matrix for a survey in which k objects are selected from a group of m

Usage

```
Amaker(m, k)
```

Arguments

m	the number of objects
k	the number of objects selected

Value

...

See Also

Tmaker, Emaker, Mmaker, Pmaker, Smaker

Examples

```
Amaker(4, 2)
```

---

bertini *Evaluate Bertini Code*

Description

Write a Bertini file, evaluate it through a back-end connection to Bertini, and bring the output back into R.

Usage

```
bertini(code, dir = tempdir(), opts = "", quiet = TRUE)
```

Arguments

code	Bertini code as either a character string or function; see examples
dir	directory to place the files in, without an ending /
opts	options for bertini
quiet	show bertini output

Value

an object of class bertini

Examples

```
## Not run:

# where does the circle intersect the line y = x?
code <- "
INPUT

variable_group x, y;
function f, g;

f = x^2 + y^2 - 1;
g = y - x;

END;
"
bertini(code)
```

```
c(sqrt(2)/2, sqrt(2)/2)



# where do the surfaces
#  x^2 - y^2 - z^2 - 1/2
#  x^2 + y^2 + z^2 - 9
#  x^2/4 + y^2/4 - z^2
# intersect?
#
code <- "
INPUT

variable_group x, y, z;
function f, g, h;

f = x^2 - y^2 - z^2 - 1/2;
g = x^2 + y^2 + z^2 - 9;
h = x^2/4 + y^2/4 - z^2;

END;
"
bertini(code)

# algebraic solution :
c(sqrt(19)/2, 7/(2*sqrt(5)), 3/sqrt(5)) # +/- each ordinate



# example from bertini manual
code <- "
INPUT

variable_group x, y;
function f, g;

f = x^2 - 1;
g = x + y - 1;

END;
"
out <- bertini(code)
summary(out)



# non zero-dimensional example
code <- "
CONFIG
  TRACKTYPE: 1;
END;

INPUT
```

```
  variable_group x, y, z;
  function f1, f2;
  f1 = x^2-y;
  f2 = x^3-z;
END;
"
out <- bertini(code)
# bertini(code, quiet = FALSE) # print broken here
```

```
## End(Not run)
```

---

bump *Convert Dimensions of Approval Data*

Description

Convert dimensions of approval data, see details for more.

Usage

```
bump(x, n, k, vin, vout, method = c("popular", "even"))
```

Arguments

x	the summary given
n	the number of objects in the running
k	the number of objects approved
vin	the level of summary given
vout	the level of summary/expectation desired
method	"popular" (default) or "even", see details

Details

In a survey in which k objects are approved from a list of n objects by N voters, the survey responses can be summarized with choose(n, k) frequencies. bump can summarize these frequencies by computing the number of votes for any lower order grouping primarily using the Tmaker function. We refer to this as "bumping down". Given a ith summary (the number of votes for each i-grouping), we can compute the expected (i+1)-group votes by evenly distributing each i-groups votes to the (i+1)-groups containing the i-groups and summing over each i-group's contribution. This is "bumping up".

See examples for the cookie example.

As a simple example of bumping up, suppose we have a survey in which 100 individuals select their favorite 3 items out of 6 items. The total number of votes cast is therefore 100*3 = 300.

If that is all that is known, then the (one) bumped up expected dataset would simply expect each of the 6 items to be listed equally frequently: a vector of length 6 with each element equal to 300/6 = 50. We would expect each of the 15 pairings to have 300/choose(6, 2) = 300 / 15 = 20 votes, and each of the 20 triples to have 300/choose(6, 3) = 5 votes.

Now suppose we learn that the six objects were voted for 30, 40, 50, 50, 60, and 70 times, respectively, but nothing more. Obviously, we could then compute 300 votes had been cast (V0 = 100 voters), but "looking up" we could also guess that the pairing [12] was voted for 30/choose(5, 1) + 40/choose(5, 1) = 14 times. The reasoning is that if object 1 were observed 30 times and 2 40 times, if each were paired evenly with each of the others 1 would contribute 30/choose(5, 1) votes to the pairing and 2 40/choose(5, 1). The choose(5, 1) corresponds to the number of pairings that 1 (or 2) is present in: [12], [13], [14], [15], [16]. The same idea can be used to estimate the number of votes for each of the choose(6, 2) = 15 pairs. (See examples.) This is bumping up; for any level of summary, we can bump it up as far as we like (all the way up to the set itself).

Bumping down is easier. The only thing needed to know is that it follows the order of the subsets function. For example, in the above voting example, the 15 pairs votes are assumed to be in the order subsets(6, 2), and the result is given in the order of subsets(6, 1).

If method = "even", exactly the above is done. If method = "popular", then when bumping up the number of votes for [12] isn't determined by each of 1 and 2 donating their samples evenly to each of their upstream pairs; but rather 1 and 2 donating to each other (as contributors of the pair [12]) according to how popular it is amongst the alternatives. In other words, 1 is thought to have (in this case) 30 votes to "give" to either 2, 3, 4, 5, or 6. If method = "even", it donates 30/5 to each. If method = "popular", it donates 40/(40+50+50+60+70) of the 30 votes to 2 (as a contributor of [12]), 50/(40+50+50+60+70) of the 30 to 3, and so on. The expected frequency of [12] is therefore made up as the sum of such contributions from each of the places the contribution might come. Here, the contributors of [12] are [1] and [2], with contributions 30*40/(40+50+50+60+70) and 40*30/(30+50+50+60+70), for a total of 9.06 expected [12] votes. The same is done for higher order votes; e.g. [123] takes even or popular contributions from [12], [13], and [14].

Value

...

Examples

```
## Not run:

V0 <- 100 # V0 = number of voters (not votes)
bump(V0, 6, 3, 0, 0) # no bump
bump(V0, 6, 3, 0, 1) # 1-up
bump(V0, 6, 3, 0, 2) # 2-up
```

```
bump(V0, 6, 3, 0, 3) # 3-up

V1 <- c(30, 40, 50, 50, 60, 70)
bump(V1, 6, 3, 1, 0) # bump down
bump(V1, 6, 3, 1, 1) # no bump
bump(V1, 6, 3, 1, 2) # 1-up
bump(V1, 6, 3, 1, 3) # 2-up

cbind(
  bump(V1, 6, 3, 1, 2, "popular"),
  bump(V1, 6, 3, 1, 2, "even")
)




data(cookie)
(out <- spectral(cookie$freq, 6, 3, cookie$cookies))

(V0 <- out$obs$V0)
bump(V0, 6, 3, 0, 0)
bump(V0, 6, 3, 0, 1)
bump(V0, 6, 3, 0, 2)
bump(V0, 6, 3, 0, 3)
out$fullExp$V0
out$decompose(out$effects[,1])

(V1 <- out$obs$V1)
bump(V1, 6, 3, 1, 0) # cbind(bump(V1, 6, 3, 1, 0), out$fullExp$V1[[1]])
bump(V1, 6, 3, 1, 1) # cbind(bump(V1, 6, 3, 1, 1), out$fullExp$V1[[2]])
bump(V1, 6, 3, 1, 2) # cbind(bump(V1, 6, 3, 1, 2), out$fullExp$V1[[3]])
bump(V1, 6, 3, 1, 3) # cbind(bump(V1, 6, 3, 1, 3), out$fullExp$V1[[4]])
out$fullExp$V1 # the sampler doesn't distribute it's samples up evenly

(V2 <- out$obs$V2)
bump(V2, 6, 3, 2, 0) # cbind(bump(V2, 6, 3, 2, 0), out$fullExp$V2[[1]])
bump(V2, 6, 3, 2, 1) # cbind(bump(V2, 6, 3, 2, 1), out$fullExp$V2[[2]])
bump(V2, 6, 3, 2, 2) # cbind(bump(V2, 6, 3, 2, 2), out$fullExp$V2[[3]])
bump(V2, 6, 3, 2, 3) # cbind(bump(V2, 6, 3, 2, 3), out$fullExp$V2[[4]])

(V3 <- out$obs$V3)
bump(V3, 6, 3, 3, 0)
bump(V3, 6, 3, 3, 1)
bump(V3, 6, 3, 3, 2)
bump(V3, 6, 3, 3, 3)




## End(Not run)
```

`city` *Living Location Preference Dataset*

Description

1439 people were asked to rank three areas in which to live, the city (1), suburbs (2), and the country (3). 637 of these individuals themselves lived in a city, 500 in the suburbs, and 302 in the country.

Usage

```
data(city)
```

Format

A 6x3 matrix

Author(s)

1972 National Opinion Research Center Amalgam Survey

References

Marden, J. I. (1995). *Analyzing and Modeling Rank Data*, London: Chapman & Hall. p.35.

`condorcet` *Find a Condorcet Choice.*

Description

Try to compute find a Condorcet choice given a full ranking of m objects.

Usage

```
condorcet(data, names)
```

Arguments

`data`	the data, a vector of counts of each permutation of the m objects (m is the length of data)
`names`	character vector of the names of the m objects

Details

In a ranking of m objects, the Condorcet choice is the choice that wins over every other choice in pairwise comparisons. See Marden (1995), p.20 for details.

Value

...

References

Marden, J. I. (1995). *Analyzing and Modeling Rank Data*, London: Chapman & Hall. p.20.

See Also

`Pmaker`

Examples

```
data(city)

condorcet(city[,"city"], colnames(city))    # among city-dwellers
condorcet(city[,"suburb"], colnames(city))  # among suburb-dwellers
condorcet(city[,"country"], colnames(city)) # among country-dwellers
condorcet(rowSums(city), colnames(city))    # overall winner
```

`cookie` *Girl Scout Cookie Preferences*

Description

A small approval dataset on Girl Scout cookies.

Usage

```
data(cookie)
```

Format

A list with elements "freq", "raw", "sorted", and "cookies"

Author(s)

Ann Johnston and Michael Orrison

References

Johnston, A. and Orrison, M. (2011) Markov Bases for Noncommutative Harmonic Analysis of Partially Ranked Data. Undergraduate Thesis, Department of Mathematics, Harvey Mudd College.

`count` *Count Integer Points in a Polytope*

Description

`count` uses LattE's count function to count the (integer) lattice points in a polytope and compute Ehrhart polynomials.

Usage

```
count(spec, dir = tempdir(), opts = "", quiet = TRUE, mpoly = TRUE)
```

Arguments

`spec`	specification, see details and examples
`dir`	directory to place the files in, without an ending /
`opts`	options for count; "" for a hyperplane representation, "–vrep" for a vertex representation; see the LattE manual at http://www.math.ucdavis.edu/~latte
`quiet`	show latte output
`mpoly`	when opts = "–ehrhart-polynomial", return the mpoly version of it

Details

The specification should be one of the following: (1) a character string or strings containing an inequality in the mpoly expression format (see examples), (2) a list of vertices, or (3) raw code for LattE's count program. If a character vector is supplied, (1) and (3) are distinguished by the number of strings.

Behind the scenes, count works by writing a latte file and running count on it. If a specification other than a length one character is given to it (which is considered to be the code), count attempts to convert it into LattE code and then run count on it.

Value

the count. if the count is a number has less than 10 digits, an integer is returned. if the number has 10 or more digits, an integer in a character string is returned. you may want to use the gmp package's as.bigz to parse it.

Examples

```
## Not run:




spec <- c("x + y <= 10", "x >= 1", "y >= 1")
count(spec)
count(spec, opts = "--dilation=10")
count(spec, opts = "--homog")

# by default, the output from LattE is in
list.files(tempdir())
list.files(tempdir(), recursive = TRUE)

# ehrhart polynomials
count(spec, opts = "--ehrhart-polynomial")
count(spec, opts = "--ehrhart-polynomial", mpoly = FALSE)

# ehrhart series (raw since mpoly can't handle rational functions)
count(spec, opts = "--ehrhart-series")

# simplified ehrhart series - not yet implemented
#count(spec, opts = "--simplified-ehrhart-polynomial")

# first 3 terms of the ehrhart series
count(spec, opts = "--ehrhart-taylor=3")

# multivariate generating function
count(spec, opts = "--multivariate-generating-function")


# the number of tables with the same marginals
data(politics)
count(c(
  "x11 + x12 == 10",
  "x21 + x22 == 10",
  "x11 + x21 == 9",
```

```
  "x12 + x22 == 11",
  "x11 >= 0", "x21 >= 0", "x12 >= 0", "x22 >= 0"
))
countTables(politics)


# by vertices
spec <- list(c(1,1),c(10,1),c(1,10),c(10,10))
count(spec)
count(spec, opts = "--vrep")

code <- "
5 3
1 -1 0
1 0 -1
1 -1 -1
0 1 0
0 0 1
"
count(code)




## End(Not run)
```

`countTables` *Count Similarly Margined Contingency Tables*

Description

Count the number of contingency tables with the same marginals as a given table.

Usage

```
countTables(table, margins = as.list(1:length(dim(table))), dir = tempdir(),
  opts = "", quiet = TRUE)
```

Arguments

`table`	the table of interest
`margins`	the margins to be fixed
`dir`	directory to place the files in, without an ending /
`opts`	options for count
`quiet`	show latte output

Details

`countTables` uses LattE's count function (via algstat's `count` function) to count the tables. In many cases, the number of such tables is enormous. In these cases, instead of giving back an integer `countTables` provides a character string with the integer in it; see examples.

Value

an integer

See Also

count

Examples

```
## Not run:


data(politics)
countTables(politics)

data(handy)
countTables(handy)

data(HairEyeColor)
eyeHairColor <- margin.table(HairEyeColor, 2:1)
countTables(eyeHairColor)

library(gmp)
as.bigz(countTables(eyeHairColor))


# notice that even tables with small cells can have
# huge fibers
data(drugs)
countTables(drugs)

countTables(eyeHairColor, quiet = FALSE)




## End(Not run)
```

---

drugs *Use of Alcohol, Cigarettes, and Marijuana by High School Students*

Description

A multi-way contingency table of alcohol, cigarettes, and marijuana use by high school students.

Usage

```
data(drugs)
```

Format

A 2x2x2 (contingency) table

Author(s)

Wright State University School of Medicine and the United Health Services in Dayton, Ohio

References

Agresti, A. (2002). *Categorical Data Analysis*, Basel: John Wiley & Sons, 2ed. (p.322)

`Emaker` *Create the expected higher-order statistics calculating matrix for approval data*

Description

Create the expected higher-order statistics calculating matrix for approval data

Usage

```
Emaker(m, vin, vout)
```

Arguments

`m`	the number of objects
`vin`	the (lower order) grouping level of the data
`vout`	the desired higher order grouping level

Value

...

See Also

`Tmaker`, `Amaker`, `Mmaker`, `Pmaker`, `Smaker`

Examples

```
Emaker(6, 0, 1)
Emaker(6, 0, 2)
Emaker(6, 0, 3)
Emaker(6, 0, 4)

Emaker(6, 1, 1)
Emaker(6, 1, 2)
Emaker(6, 1, 3)
Emaker(6, 1, 4)
Emaker(6, 1, 5)
Emaker(6, 1, 6)

# compare to Tmaker
Emaker(6, 1, 3) # contributors when bumping up from 1-groups to 3-groups
Tmaker(6, 3, 1)
```

haberman *Haberman's Positive Margin no Three-way Interaction MLE*

Description

Haberman's example of a log-linear model (the no-three way interaction model) that lacks an MLE even though it has positive margins.

Usage

```
data(haberman)
```

Format

A 2x2x2 (contingency) table

Author(s)

unknown

References

Haberman, S. (1974). *The Analysis of Frequency Data* 1, 2. University of Chicago Press, Chicago, IL.

handy *Handedness Data*

Description

A small fictional dataset on handedness.

Usage

```
data(handy)
```

Format

A 2x2 (contingency) table

Author(s)

David Kahle

`hierarchical` *Fitting Hierarchical Log-linear Models with Algebraic Methods*

Description

Run the Metropolis-Hastings algorithm using a Markov basis computed with 4ti2 to sample from the conditional distribution of the data given the sufficient statistics of a hierarchical model.

Usage

```
hierarchical(formula, data, iter = 10000, burn = 1000, thin = 10,
  engine = c("Cpp", "R"), method = c("ipf", "mcmc"), moves)
```

Arguments

`formula`	formula for the hierarchical log-linear model
`data`	data, typically as a table but can be in different formats. see `teshape`
`iter`	number of chain iterations
`burn`	burn-in
`thin`	thinning
`engine`	C++ or R? (C++ yields roughly a 20-25x speedup)
`method`	should the expected value (exp) be fit using iterative proportional fitting (via loglin) or the MCMC as the average of the steps?
`moves`	the markov moves for the mcmc

Details

Hierarchical fits and tests a hierarchical log-linear model on a contingency table. In many ways, hierarchical is like stats::loglin or MASS::loglm; however, there are a few key differences in the functionality of hierarchical.

The first difference is methodological. The tests conducted with hierarchical are exact tests based on the conditional distribution of the data given the sufficient statistics for the model. In other words, they are Fisher's exact test analogues for log-linear models. These tests are made possible by advances in algebraic statistics; see the first and second references below. In particular, hierarchical leverages Markov bases through the software 4ti2 to construct a Metropolis-Hastings algorithm to sample from the conditional distribution of the table given the sufficient statistics.

A second way that hierarchical differs from stats::loglin or MASS::loglm is in generalizing the kinds of tests performed. While those allow for the asymptotic unconditional testing using Pearson's X^2 test and the likelihood ratio test (MASS::loglm is simply a wrapper for stats::loglin), hierarchical gives several test statistics: Pearson's X^2, the likelihood ratio G^2, Freeman-Tukey, Cressie-Read (lambda = 2/3), and Neyman's modified X^2., see the last reference. In other words, to compute the exact p-value, iter = 1e4 samples are sampled from the conditional distribution of the table given the sufficient statistics, and then the proportion of tables that have X^2, G^2, etc. values greater than or equal to that of the observed table is the p value for the (conditional) exact test. A similar, but perhaps preferable approach, simply adds up the probabilities of the tables that have probabilities less than or equal to that of the observed table; this is the first line output in hierarchical and does not use a test statistic.

Some authors (see the third reference) suggest that for discrete problems, a "mid p value" is preferable to the traditional p value, and when presented should be interepreted in the same way. If the p value is defined to be, say, P(samps >= obs), the mid p value is defined to be P(samps > obs) + P(samps == obs)/2. The mid p value is computed for each test.

Since the tests make use of Monte Carlo sampling, standard errors (SE) are reported for each statistic. For the test statistics, this is just the standard deviation of the samples divided by the square root of the sample size, iter; they are computed and returned by the print method. The standard errors of the p values use the CLT asymptotic approximation and, therefore, warrant greater consideration when the p value is close to 0 or 1.

Value

a list containing named elements

- `steps`: an integer matrix whose columns represent individual samples from the mcmc.
- `moves`: the moves used for the proposal distribution in the mcmc, computed with 4ti2 (note that only the positive moves are given).
- `acceptProb`: the average acceptance probability of the moves, including the thinned moves.
- `param`: the fitted parameters of the log linear model.
- `df`: parameters per term in the model
- `quality`: model selection statistics AIC, AICc, and BIC.
- `residuals`: the (unstandardized) pearson residuals (O - E) / sqrt(E)
- `call`: the call.
- `obs`: the contingency table given.
- `exp`: the fit contingency table as an integer array.
- `A`: the sufficient statistics computing matrix (from Tmaker).
- `p.value`: the exact p-values of individual tests, accurate to Monte-Carlo error. these are computed as the proportion of samples with statistics equal to or larger than the oberved statistic.
- `mid.p.value`: the mid p.values, see Agresti pp.20–21.
- `statistic`: the pearson's chi-squared (X2), likelihood ratio (G2), Freeman-Tukey (FT), Cressie-Read (CR), and Neyman modified chi-squared (NM) statistics computed for the table given.
- `sampsStats`: the statistics computed for each mcmc sample.
- `cells`: the number of cells in the table.
- `method`: the method used to estimate the table.

Author(s)

David Kahle

References

Diaconis, P. and B. Sturmfels (1998). Algebraic Algorithms for Sampling from Conditional Distributions. *The Annals of Statistics* 26(1), pp.363-397.

Drton, M., B. Sturmfels, and S. Sullivant (2009). *Lectures on Algebraic Statistics*, Basel: Birkhauser Verlag AG.

Agresti, A. (2002). *Categorical Data Analysis*, Basel: John Wiley & Sons, 2ed.

Agresti, A. (1992). A Survey of Exact Inference for Contingency Tables *Statistical Science* 7(1), pp.131-153.

Read, T. and Cressie, N. (1998). *Goodness-of-Fit Statistics for Discrete Multivariate Data*, Springer-Verlag.

See Also

loglin, loglm, metropolis

Examples

```
## Not run:


## handedness introductory example
############################################################

data(handy)

(out <- hierarchical(~ Gender + Handedness, data = handy))

# hierarchical performs the same tasks as loglin and loglm,
# but hierarchical gives the exact test p values and more statistics
statsFit <- stats::loglin(handy, list(c(1),c(2)), fit = TRUE, param = TRUE)
massFit <- MASS::loglm(~ Gender + Handedness, data = handy)
# loglm is just a wrapper of loglin









# comparisons between hierarchical and loglin
############################################################

# the expected table given the sufficient statistics can be computed
# via two methods, iterative proportional fitting, and the mcmc itself:
out$exp # ipf
hierarchical(~ Gender + Handedness, data = handy, method = "mcmc")$exp
statsFit$fit # the equivalent in loglin; this is used by default in hierarchical




# the parameter values of the loglinear model can be accessed
out$param
statsFit$param
```

```
# the p-value for the overall model is available as well
# hierarchical gives the exact conditional p-value
# (conditional on the sufficient statistics)
# the five numbers correspond the probability of tables that are
# "more weird" than the observed table, where "more weird" is determined
# by having a larger X2 value (or G2, FT, CR, or NM)
out$p.value
fisher.test(handy)$p.value # out$p.value["X2"] is accurate to monte carlo error


# loglin gives the p-values using the unconditional asymptotic distributions
c(
  "X2" = pchisq(statsFit$pearson, df = statsFit$df, lower.tail = FALSE),
  "G2" = pchisq(statsFit$lrt, df = statsFit$df, lower.tail = FALSE)
)

out$mid.p.value # the mid (exact conditional) p-value is also available




# the test statistics based on the observed table and the expected
# table under the model are available
out$statistic
c(statsFit$pearson, statsFit$lrt) # loglin only gives X2 and G2




# the markov basis used for the proposal distribution of the metropolis-hastings
# algorithm are returned. the proposal distribution is uniform on +/-
# the moves added to the current table
out$moves
# they are easier understood as tables
vec2tab(out$moves, dim(handy))
# notice that the marginals stay fixed:
handy + vec2tab(out$moves, dim(handy))




# these were computed as the markov basis of the integer matrix
out$A
markov(out$A)
out$moves




# the moves are also sometimes written in tableau form (LAS p.13)
tableau(out$moves, dim(handy))
# that's +1 the the table in elements [1,1] and [2,2]
# and -1 in the table in elements [1,2] and [2,1]
```

```
# the acceptance probability of the MCMC is retained
out$acceptProb



# various model assessment measures are also available
out$quality



# the number of independent parameters per term are in df
out$df



# as an added help, you may find the visuals in vcd useful:
# library(vcd)
# mosaic(~ Gender + Handedness, data = handy, shade = TRUE, legend = TRUE)
```

```
## politics example - with computing the exact p value by hand
############################################################

data(politics)

(out <- hierarchical(~ Personality + Party, data = politics))
statsFit <- stats::loglin(politics, as.list(1:2), fit = TRUE, param = TRUE)

out$p.value
# exact without monte-carlo error
sum(dhyper(c(0:3,6:9), 10, 10, 9))
fisher.test(politics)$p.value
```

```
round(dhyper(0:9, 10, 10, 9), 4)


# comparisons :
out$exp
statsFit$fit

out$param
statsFit$param

out$p.value # exact
c(
  "X2" = pchisq(statsFit$pearson, df = statsFit$df, lower.tail = FALSE),
  "G2" = pchisq(statsFit$lrt, df = statsFit$df, lower.tail = FALSE)
) # asymptotic approximation
fisher.test(politics)$p.value # accurate to monte carlo error

out$statistic # accurate to monte carlo error
c(statsFit$pearson, statsFit$lrt)

# mosaic(~ Personality + Party, data = politics, shade = TRUE, legend = TRUE)









## eyeHairColor from the Diaconis and Sturmfels reference
################################################################

data(HairEyeColor)
eyeHairColor <- margin.table(HairEyeColor, 2:1)

outC <- hierarchical(~ Eye + Hair, data = eyeHairColor)
outR <- hierarchical(~ Eye + Hair, data = eyeHairColor, engine = "R")

# doesn't work even with workspace = 2E9 (with over 4.5Gb in memory)
#fisher.test(eyeHairColor, hybrid = TRUE, workspace = 2E9)

tableau(outC$moves, dim(eyeHairColor))


# library(microbenchmark)
# microbenchmark(
#   hierarchical(~ Eye + Hair, data = eyeHairColor),
#   hierarchical(~ Eye + Hair, data = eyeHairColor, engine = "R")
# )
# 5-10 times faster; much faster with increased iter


# mosaic(~ Eye + Hair, data = HairEyeColor, shade = TRUE, legend = TRUE)
```

```
## abortion preference example from the
## Diaconis and Sturmfels reference pp. 379--381
## a no 3-way interaction model
############################################################

data(abortion)

out <- hierarchical(
  ~ Education*Abortion + Abortion*Denomination + Education*Denomination,
  data = abortion,
  iter = 10000, burn = 50000, thin = 50
)
out$p.value


vec2tab(rowMeans(out$steps), dim(abortion)) # cf. p. 380
loglin(abortion, subsets(1:3, 2), fit = TRUE)$fit



out$param
loglin(abortion, subsets(1:3, 2), param = TRUE)$param



qqplot(rchisq(1055, df = 8), out$sampsStats$X2s)
curve(1*x, from = 0, to = 30, add = TRUE, col = "red")

( nMoves <- 2*ncol(out$moves) ) # DS uses 110
# the markov basis is larger than it needs to be









## loglin no three-way interaction model example
############################################################

# the help for fits the no three-way interaction model on HairEyeColor,
# finds a .66196 p-value using the asymptotic distribution, and concludes
# a good fit:
data(HairEyeColor)

fit <- loglin(HairEyeColor, subsets(1:3, 2), fit = TRUE, param = TRUE)
mod <- hierarchical(~ Eye*Hair + Hair*Sex + Eye*Sex, data = HairEyeColor)
```

```
# p values
pchisq(fit$lrt, fit$df, lower.tail = FALSE) # see ?loglin
mod$p.value

# test statistics
c(fit$pearson, fit$lrt)
mod$statistic

# fits (estimated tables)
fit$fit
mod$exp
mod$obs


# checking the autocorrelation
acf(mod$sampsStats$PRs)
mod <- hierarchical(~ Eye*Hair + Hair*Sex + Eye*Sex, data = HairEyeColor, thin = 100)
acf(mod$sampsStats$PRs) # got it!


# the slight differences in fit$fit and mod$exp (both done with ipf from loglin)
# are due to differences in variable order:
loglin(HairEyeColor, subsets(1:3, 2), fit = TRUE)$fit
loglin(HairEyeColor, subsets(1:3, 2)[c(1,3,2)], fit = TRUE)$fit

# a few model moves
vec2tab(mod$moves[,1], dim(HairEyeColor))
vec2tab(mod$moves[,50], dim(HairEyeColor))
-vec2tab(mod$moves[,50], dim(HairEyeColor))

# they contribute 0 to the marginals of the table
vec2tab(mod$moves[,50], dim(HairEyeColor))
mod$A %*% mod$move[,50]
vec2tab(mod$A %*% mod$move[,50], dim(HairEyeColor))

HairEyeColor
HairEyeColor + vec2tab(mod$moves[,50], dim(HairEyeColor))







## a table with positive marginals but no MLE for
## the no-three way interaction model
############################################################


data(haberman)

mod <- hierarchical(~ X1*X2 + X2*X3 + X1*X3, data = haberman)

statsFit <- loglin(haberman, subsets(1:3, 2), param = TRUE, fit = TRUE)
```

```
statsFit$fit
statsFit$param
c(statsFit$pearson, statsFit$lrt)

algstatFit <- hierarchical(~ X1*X2 + X2*X3 + X1*X3, data = haberman, method = "mcmc")
algstatFit$exp
algstatFit$param
algstatFit$statistic









## an example from agresti, p.322
############################################################

data(drugs)
ftable(aperm(drugs, c(3, 1, 2))) # = table 8.3

out <- hierarchical(~Alcohol + Cigarette + Marijuana, data = drugs)
matrix(round(aperm(out$exp, c(2,1,3)), 1), byrow = FALSE)

loglin(drugs, as.list(1:3), fit = TRUE)$fit
loglin(drugs, as.list(1:3), param = TRUE)$param

# # the saturated model issues a warning from markov, but works :
# out <- hierarchical(~Alcohol * Cigarette * Marijuana, data = drugs)
# matrix(round(aperm(out$exp, c(2,1,3)), 1), byrow = FALSE)


ftable(aperm(out$exp, c(3,1,2)))

stats <- loglin(drugs, as.list(1:3), fit = TRUE, param = TRUE)


# considered via glm

df <- as.data.frame(drugs)
mod <- glm(Freq ~ Alcohol + Cigarette + Marijuana, data = df, family = poisson)
summary(mod)
mod$fitted.values


# the same can be done with glm :

mod <- glm(
  Freq ~ Alcohol + Cigarette + Marijuana,
  data = as.data.frame(drugs), family = poisson
)
summary(mod)
```

```
matrix(round(mod$fitted.values[c(1,3,2,4,5,7,6,8)],1))



mod <- glm(
  Freq ~ Alcohol * Cigarette + Marijuana,
  data = as.data.frame(drugs), family = poisson
)
summary(mod)
matrix(round(mod$fitted.values[c(1,3,2,4,5,7,6,8)],1))


mod <- glm(
  Freq ~ Alcohol * Cigarette * Marijuana,
  data = as.data.frame(drugs), family = poisson
)
summary(mod)
matrix(round(mod$fitted.values[c(1,3,2,4,5,7,6,8)],1))









## End(Not run)
```

---

hmat *Construct a Hierarchical Model Matrix*

Description

Determine the A matrix associated with a hierarchical model on a contingency table. In algebraic statistics, the A matrix of a log-linear model is the transpose of the design matrix of the (cell-means parameterized) ANOVA corresponding to the model.

Usage

```
hmat(varlvls, facets)
```

Arguments

varlvls	a vector containing the number of levels of each variable
facets	the facets generating the hierarchical model, a list of vectors of variable indices

Value

a named matrix

References

Drton, M., B. Sturmfels, and S. Sullivant (2009). *Lectures on Algebraic Statistics*, Basel: Birkhauser Verlag AG.

Examples

```
# LAS example 1.2.11, p.16
varlvls <- c(2,2,2,2)
facets <- list(c(1,2), c(1,4), c(2,3))
( A <- hmat(varlvls, facets) )

# 2x2 independence example
# following convention, the first index indicates rows
varlvls <- c(2,2)
facets <- list(1,2)
( A <- hmat(varlvls, facets) )

printForMarkov <- function(A){
  cat(paste(nrow(A), ncol(A)))
  cat("\n")
  cat(apply(unname(A), 1, paste, collapse = " "), sep = "\n")
  cat("\n")
}
printForMarkov(A)
```

---

is.bertini *Bertini Object Check*

Description

Test whether an object is an bertini object.

Usage

```
is.bertini(x)
```

Arguments

x	object to be tested

Value

Vector of logicals.

Examples

```
# see ?bertini
```

is.linear *Test whether an mpoly object is linear.*

Description

Test whether an mpoly object is linear.

Usage

```
is.linear(x)
```

Arguments

x	an mpoly or mpolyList object

Value

a logical vector

Examples

```
## Not run:

is.linear(mp("0"))
is.linear(mp("x + 1"))
is.linear(mp("x + y"))
is.linear(mp(c("0", "x + y")))

is.linear(mp("x + x y"))
is.linear(mp(c("x + x y", "x")))



## End(Not run)
```

is.m2 *Macaulay2 Object Check*

Description

Test whether an object is an m2 object.

Usage

```
is.m2(x)
```

Arguments

x	object to be tested

Value

Vector of logicals.

Examples

```
## Not run:

is.m2(m2("13^1000"))


## End(Not run)
```

---

kprod *Iterated Kronecker product*

Description

Compute the Kronecker product of several matrices.

Usage

```
kprod(...)
```

Arguments

...	a listing of matrices

Details

If kronecker is the function that computes A x B, kprod computes A x B x C and so on; it's a wrapper of Reduce and kronecker.

Value

... a matrix that is the kronecker product of those matrices (from left to right)

Examples

```
kprod(diag(2), t(ones(2)))
kprod(t(ones(2)), diag(2))

kprod(diag(2), t(ones(2)), t(ones(2)))
kprod(t(ones(2)), diag(2), t(ones(2)))
kprod(t(ones(2)), t(ones(2)), diag(2))


rbind(
  kprod(diag(2), t(ones(2))),
  kprod(t(ones(2)), diag(2))
)
```

latteMax *Solve a Linear Program (Maximization)*

Description

latteMax uses LattE's maximize function to find the maximum of a linear objective function over the integers satisfying linearity constraints. This makes use of the digging algorithm; see the LattE manual at http://www.math.ucdavis.edu/~latte for details.

Usage

```
latteMax(objective, constraints, method = c("lp", "cones"), dir = tempdir(),
  opts = "", quiet = TRUE)
```

Arguments

objective	a linear polynomial to pass to mp, see examples
constraints	a collection of linear polynomial (in)equalities that define the feasibility region, the integers in the polytope
method	method LP or cones
dir	directory to place the files in, without an ending /
opts	options; see the LattE manual at http://www.math.ucdavis.edu/~latte
quiet	show latte output

Value

the count. if the count is a number has less than 10 digits, an integer is returned. if the number has 10 or more digits, an integer in a character string is returned. you may want to use the gmp package's as.bigz to parse it.

Examples

```
## Not run:

latteMax("-2 x + 3 y", c("x + y <= 10", "x >= 0", "y >= 0"))


df <- expand.grid(x = 0:10, y = 0:10)
df <- subset(df, x + y <= 10)
df$val <- apply(df, 1, function(v) -2*v[1] + 3*v[2])
df[which.max(df$val),]

library(ggplot2)
qplot(x, y, data = df, size = val)



## End(Not run)
```

latteMin *Solve a Linear Program (Minimization)*

Description

latteMin uses LattE's minimize function to find the minimum of a linear objective function over the integers satisfying linearity constraints. This makes use of the digging algorithm; see the LattE manual at http://www.math.ucdavis.edu/~latte for details.

Usage

```
latteMin(objective, constraints, method = c("lp", "cones"), dir = tempdir(),
  opts = "", quiet = TRUE)
```

Arguments

objective	a linear polynomial to pass to mp, see examples
constraints	a collection of linear polynomial (in)equalities that define the feasibility region, the integers in the polytope
method	method LP or cones
dir	directory to place the files in, without an ending /
opts	options; see the LattE manual at http://www.math.ucdavis.edu/~latte
quiet	show latte output

Value

the count. if the count is a number has less than 10 digits, an integer is returned. if the number has 10 or more digits, an integer in a character string is returned. you may want to use the gmp package's as.bigz to parse it.

Examples

```
## Not run:

latteMin("-2 x + 3 y", c("x + y <= 10", "x >= 0", "y >= 0"))
latteMin("-2 x + 3 y", c("x + y <= 10", "x >= 0", "y >= 0"),
  method = "cones") # ??


df <- expand.grid(x = 0:10, y = 0:10)
df <- subset(df, x + y <= 10)
df$val <- apply(df, 1, function(v) -2*v[1] + 3*v[2])
df[which.min(df$val),]

library(ggplot2)
qplot(x, y, data = df, size = val)
```

```
latteMin("-2 x - 3 y - 4 z", c(
  "3 x + 2 y + z <= 10",
  "2 x + 5 y + 3 z <= 15",
  "x >= 0", "y >= 0", "z >= 0"
), "cones",quiet = FALSE)

df <- expand.grid(x = 0:10, y = 0:10, z = 0:10)
df <- subset(df,
  (3*x + 2*y + 1*z <= 10) &
  (2*x + 5*y + 3*z <= 15)
)

df$val <- apply(df, 1, function(v) -2*v[1] + -3*v[2] + -4*v[3])
df[which.min(df$val),]




## End(Not run)
```

`lower` *Create a lower triangular matrix*

Description

Create a lower triangular matrix.

Usage

```
lower(x)
```

Arguments

x	a vector

Value

...

See Also

upper

Examples

```
upper(1:3)
lower(1:3)

upper(1:6)
lower(1:6)

upper(rnorm(6))
```

`lpnorm` *Lp Norm*

Description

Compute the Lp norm of a vector.

Usage

```
lpnorm(x, p = 2)
```

Arguments

x	x
p	p

Value

...

Examples

```
lpnorm(1:10)
lpnorm(matrix(1:25, 5, 5))
lpnorm(split(1:25, rep(1:5, each = 5)))

lpnorm(1:10, 1)
lpnorm(matrix(1:25, 5, 5), 1)
lpnorm(split(1:25, rep(1:5, each = 5)), 1)

lpnorm(rnorm(10), 0)
lpnorm(matrix(rnorm(25), 5, 5), 0)
lpnorm(split(rnorm(25), rep(1:5, each = 5)), 0)

lpnorm(-5:5, Inf)
lpnorm(matrix(-25:-1, 5, 5), Inf)
lpnorm(split(-25:-1, rep(1:5, each = 5)), Inf)
```

m2 *Evaluate Macaulay2 Code*

Description

Write a Macaulay2 file, evaluate it through a back-end connection to Macaulay2, and bring the output back into R.

Usage

```
m2(code, dir = tempdir(), opts = "--script")
```

Arguments

code	Macaulay2 code as either a character string or function; see examples
dir	directory to place the files in
opts	options for m2

Value

an object of class m2

Examples

```
## Not run:

options(digits = 20)
13^20
m2("13^20") # correct answer
m2("toRR(20,(19004963774880800571392-13^20)/13^20)") # relative error
options(digits = 7)

code <- "
1+1
2+3
100!
R = QQ[x,y,z]
(x+y)^10
curve = ideal( x^4-y^5, x^3-y^7 )
gens gb curve
m = matrix {{x^2, x^2-y^2, x*y*z^7 }}
image m
R = QQ[a..d]
I = ideal(a^3-b^2*c, b*c^2-c*d^2, c^3)
G = gens gb I
G
"
m2(code)


code <- "
R = QQ[x,y,z,t]
I = ideal( t^4 - x, t^3 - y, t^2 - z)
```

```
gens gb I
"
m2(code)


## End(Not run)
```

---

markov *Compute a Markov Basis with 4ti2*

Description

A Markov basis of a matrix A is computed with the markov function of 4ti2, obtained with the LattE-integrale bundle.

Usage

```
markov(mat, format = c("mat", "vec", "tab"), dim = NULL, all = FALSE,
  dir = tempdir(), opts = "-parb", quiet = TRUE, dbName)
```

Arguments

mat	a matrix; for example the output of hmat
format	how the moves should be returned (if "mat", moves are columns)
dim	the dimension to be used in vec2tab if format = "tab" is used, oftentimes a vector of the number of levels of each variable in order
all	if TRUE, all moves (+ and -) are given. if FALSE, only the + moves are given.
dir	directory to place the files in, without an ending /
opts	options for markov
quiet	show 4ti2 output
dbName	the name of the model in the markov bases database, http://markov-bases.de, see examples

Value

a matrix containing the Markov basis as its columns (for easy addition to tables)

References

Drton, M., B. Sturmfels, and S. Sullivant (2009). *Lectures on Algebraic Statistics*, Basel: Birkhauser Verlag AG.

Examples

```
## Not run:



# 2x2 independence example
# following convention, the first index indicates rows
```

```
varlvls <- c(2,2)
facets <- list(1,2)
( A <- hmat(varlvls, facets) )
markov(A)
markov(A, "vec")
markov(A, "tab", varlvls)
markov(A, "tab", varlvls, TRUE)



# 3x3 independence example
# following convention, the first index indicates rows
varlvls <- c(3,3)
facets <- list(1,2)
( A <- hmat(varlvls, facets) )
markov(A)
markov(A, "vec")
markov(A, "tab", varlvls)
markov(A, "tab", varlvls, TRUE)



# LAS example 1.2.1, p.12 (2x3 independence)
varlvls <- c(2,3)
facets <- list(1, 2)
( A <- hmat(varlvls, facets) )
markov(A, "tab", varlvls)
# Prop 1.2.2 says that there should be
2*choose(2, 2)*choose(3,2) # = 6
# moves.
markov(A, "tab", varlvls, TRUE)



# LAS example 1.2.12, p.17  (no 3-way interaction)
varlvls <- c(2,2,2)
facets <- list(c(1,2), c(1,3), c(2,3))
( A <- hmat(varlvls, facets) )
markov(A)



# LAS example 1.2.12, p.16  (no 3-way interaction)
varlvls <- c(2,2,2,2)
facets <- list(c(1,2), c(1,4), c(2,3))
( A <- hmat(varlvls, facets) )
markov(A)
markov(A, "tab", varlvls) # hard to understand
tableau(markov(A), varlvls)
```

```
# using the markov bases database, must be connected to internet
# A <- markov(dbName = "ind3-3")
B <- markov(hmat(c(3,3), list(1,2)))
# all(A == B)
```

```
markov(diag(1, 10))


## End(Not run)
```

---

mchoose *Multinomial Coefficient*

Description

Compute the multinomial coefficient.

Usage

```
mchoose(n, x)
```

Arguments

n	an integer
x	a vector of integers

Details

This function computes the multinomial coefficient by computing the factorial of each number on a log scale, differencing log(n!) - sum(log(x!)), and then exponentiating. It then checks to see if this is an integer; if it's not, it issues a warning.

Value

...

Examples

```
mchoose(6, c(2,2,1,1))
```

`metropolis` *Markov Basis Metropolis-Hastings Algorithm*

Description

Given a starting table (as a vector) and a loglinear model matrix A, compute the Markov basis of A with 4ti2 and then run the Metropolis-Hastings algorithm starting with the starting table.

Usage

```
metropolis(init, moves, iter = 1000, burn = 1000, thin = 10,
  engine = c("Cpp", "R"))
```

Arguments

`init`	the initial step
`moves`	the markov basis (the negatives will be added). see ?markov
`iter`	number of chain iterations
`burn`	burn-in
`thin`	thinning
`engine`	C++ or R? (C++ yields roughly a 20-25x speedup)

Details

See Algorithm 1.1.13 in LAS, the reference below.

Value

a list

Author(s)

David Kahle

References

Drton, M., B. Sturmfels, and S. Sullivant (2009). *Lectures on Algebraic Statistics*, Basel: Birkhauser Verlag AG.

Examples

```
## Not run:



data(handy)

exp   <- loglin(handy, as.list(1:2), fit = TRUE)$fit
e <- unname(tab2vec(exp))
h <- t(t(unname(tab2vec(handy))))
chisq <- algstat:::computeChisqsCpp(h, e)

out <- hierarchical(~ Gender + Handedness, data = handy)
chisqs <- algstat:::computeChisqsCpp(out$steps, e)

mean(chisqs >= chisq)
fisher.test(handy)$p.value




A <- hmat(c(2,2), as.list(1:2))
moves <- markov(A)
outC <- metropolis(tab2vec(handy), moves, 1e4, engine = "Cpp")
str(outC)
outR <- metropolis(tab2vec(handy), moves, 1e4, engine = "R", thin = 20)
str(outR)

# showSteps(out$steps)


library(microbenchmark)
microbenchmark(
  metropolis(tab2vec(handy), moves, engine = "Cpp"),
  metropolis(tab2vec(handy), moves, engine = "R")
)

# cpp ~ 20-25x faster









showSteps <- function(steps){
  apply(steps, 2, function(x){
    x <- format(x)
    tab <- vec2tab(x, dim(handy))
```

```
    message(
      paste(
        apply(tab, 1, paste, collapse = " "),
        collapse = " "
      )
    )
    message("
", appendLF = F)
  })
  invisible()
}
# showSteps(out$steps)
```

```
## End(Not run)
```

`Mmaker` *Marginals matrix*

Description

Compute the marginals matrix for a full ranking of m objects

Usage

```
Mmaker(m)
```

Arguments

`m`	the number of objects

Details

This is the transpose of the marginals matrix presented in Marden (1995).

Value

...

References

Marden, J. I. (1995). *Analyzing and Modeling Rank Data*, London: Chapman & Hall. p.42.

See Also

`Tmaker`, `Amaker`, `Emaker`, `Pmaker`, `Smaker`

Examples

```
data(city)

Mmaker(3)
Mmaker(3) %*% city
```

`ones` *Ones Vector*

Description

Make a column vector of ones.

Usage

```
ones(n)
```

Arguments

`n`	how many ones

Value

a column vector of ones as an integer matrix

Examples

```
ones(5)
str(ones(5))
```

`Pmaker` *Pairs matrix*

Description

Compute the pairs matrix for a full ranking of m objects

Usage

```
Pmaker(m)
```

Arguments

`m`	the number of objects

Details

This is the transpose of the pairs matrix presented in Marden (1995).

Value

...

References

Marden, J. I. (1995). *Analyzing and Modeling Rank Data*, London: Chapman & Hall. p.42.

See Also

Tmaker, Amaker, Emaker, Mmaker, Smaker

Examples

```
data(city)

Pmaker(3)
Pmaker(3) %*% city
# 1 = city, 2 = suburb, 3 = country

# looking just among city folk, generate the pairs matrix
city[,"city",drop=FALSE] # the data
m <- sum(city[,"city"])
k <- (Pmaker(3) %*% city)[,1]
Khat <- upper(k) + lower(m-k)
colnames(Khat) <- row.names(Khat) <- colnames(city)
Khat
round(Khat / m, 2) # % times row is rated over column


# worked out: city is voted over suburb in 123 , 132, and 231, equaling
210 + 23 + 8   # = Khat[1,2]
# whereas suburb is rated over city in 213, 312, 321, equaling
111 + 204 + 81 # = Khat[2,1]


# is there a condorcet choice?

p <- ncol(Khat)
Khat[which(diag(p) == 1)] <- NA
K2 <- t(apply(Khat, 1, function(v) v[!is.na(v)])) # remove diag elts
boole <- apply(K2/m, 1, function(x) all(x > .5))
if(any(boole)) names(boole)[which(boole)]
# suburb is a condorcet choice
```

---

politicalGoals *Relative Rankings of Importance of Four Political Goals*

Description

2262 Germans were asked to rank the following in order of importance : (1) maintain order in the nation, "order"; (2) give people more say in decisions in the government, "say"; (3) fight rising prices, "prices"; and (4) protect freedom of speech, "freedom".

Usage

```
data(politicalGoals)
```

Format

An atomic named vector of length 4! = 24

Author(s)

The first reference

References

Barnes, S. H. and Kaase, M. (1979). *Political Action: Mass Participation in Five Western Countries*. Sage, Beverly Hills, CA.

Croon, M. A. (1998). Latent class models for the analysis of rankings. *New Developments in Psychological Choice Modeling*, 99–121. Feger, Klauer, and de Soete, eds. North-Holland, Amsterdam.

---

politics *Politics by Personality*

Description

A 2-by-2 contingency table comparing political identification and personality.

Usage

```
data(politics)
```

Format

A 2x2 (contingency) table

Author(s)

David Kahle, simplifying the dataset found on p.622 of Sheskin (see References)

References

Sheskin, D. J. Handbook of Parametric and Nonparametric Statistical Procedures. 4ed. Chapman and Hall/CRC Press, 2007.

polyOptim *Polynomial Optimization*

Description

Find the collection of critical points of a multivariate polynomial unconstrained or constrained to an affine variety (algebraic set; solution set of multivariate polynomials).

Usage

```
polyOptim(objective, constraints, varOrder, ...)
```

Arguments

objective	the objective polynomial (as a character or mpoly)
constraints	(as a character or mpoly/mpolyList)
varOrder	variable order (see examples)
...	stuff to pass to bertini

Value

an object of class bertini

Examples

```
## Not run:

# unconstrained optimization of polynomial functions is available
polyOptim("x^2")
polyOptim("-x^2")
polyOptim("-(x - 2)^2")
polyOptim("-(x^2 + y^2)")
polyOptim("-(x^2 + (y - 2)^2)")

polyOptim("(x - 1) (x - 2) (x - 3)") # fix global labeling


# constrained optimization over the affine varieties is also available
# (affine variety = solution set of polynomial equations)

# find the critical points of the plane f(x,y) = x + y
# over the unit circle x^2 + y^2 = 1
polyOptim("x + y", "x^2 + y^2 = 1")

# you can specify them as a combo of mpoly, mpolyList, and characters
o <- mp("x + y")
c <- "x^2 + y^2 = 1"
polyOptim(o, c)

c <- mp("x^2 + y^2 - 1")
polyOptim(o, c)

out <- polyOptim("x + y", c)
```

```
str(out)

# another example, note the solutions are computed over the complex numbers
polyOptim("x^2 y", "x^2 + y^2 = 3")
# solutions: (+-sqrt(2), +-1) and (0, +-sqrt(3))




## End(Not run)
```

`polySolve` *Solve a System of Polynomial Equations*

Description

`polySolve` solves a system of polynomial equations, specifiable in any of several ways.

Usage

```
polySolve(lhs, rhs, varOrder, ...)
```

Arguments

`lhs`	a mpolyList or character vector of left hand sides
`rhs`	a mpolyList or character vector of right hand sides
`varOrder`	variable order (see examples)
`...`	stuff to pass to bertini

Value

an object of class bertini

See Also

`variety`, `bertini`

Examples

```
## Not run:

# it can solve linear systems
# (here where the line y = x intersects y = 2 - x)
polySolve(c("y", "y"), c("x", "2 - x"), c("x", "y"))

# or nonlinear systems
polySolve(c("y", "y"), c("x^2", "2 - x^2"), c("x", "y"))

# perhaps an easier specification is equations themselves
# with either the " = " or " == " specifications
# varOrder is used to order the solutions returned
polySolve(c("y = x^2", "y = 2 - x^2"), varOrder = c("x", "y"))
```

```
polySolve(c("y == x^2", "y == 2 - x^2"), varOrder = c("x", "y"))


# mpoly objects can be given instead of character strings
lhs <- mp(c("y - (2 - x)", "x y"))
rhs <- mp(c("0","0"))
polySolve(lhs, rhs, varOrder = c("x", "y"))

# if no default right hand side is given, and no "=" or "==" is found,
# rhs is taken to be 0's.
# below is where the lines y = x and y = -x intersect the unit circle
polySolve(c("(y - x) (y + x)", "x^2 + y^2 - 1"))

# the output object is a bertini object
out <- polySolve(c("(y - x) (y + x)", "x^2 + y^2 - 1"))
str(out,1)

# here is the code that was run :
cat(out$bertiniCode)

# the finite and real solutions:
out$finite_solutions
out$real_finite_solutions




# example from Riccomagno (2008), p. 399
polySolve(c(
  "x (x - 2) (x - 4) (x - 3)",
  "(y - 4) (y - 2) y",
  "(y - 2) (x + y - 4)",
  "(x - 3) (x + y - 4)"
))


## End(Not run)
```

`print.bertini` *Pretty Printing of Bertini Output*

Description

Pretty printing of Bertini output.

Usage

```
## S3 method for class 'bertini'
print(x, digits = 3, ...)
```

Arguments

`x`	an object of class bertini
`digits`	digits to round to
`...`	additional parameters

Value

Invisible string of the printed object.

Examples

```
## Not run:

# see ?bertini

variety("x^2 + 1")
variety(c("x^2 + 1 + y","y"))


## End(Not run)
```

`print.hierarchical` *Pretty Printing of Hierarchical's Output*

Description

Pretty printing of hierarchical's output.

Usage

```
## S3 method for class 'hierarchical'
print(x, digits = 4, ...)
```

Arguments

`x`	an object of class hierarchical
`digits`	digits to round to
`...`	additional parameters

Value

Invisible string of the printed object.

Examples

```
# see ?hierarchical
```

print.m2 *Pretty printing of Macaulay2 output.*

Description

Pretty printing of Macaulay2 output.

Usage

```
## S3 method for class 'm2'
print(x, ...)
```

Arguments

x	an object of class m2
...	additional parameters

Value

Invisible string of the printed object.

Examples

```
## Not run:

m2("13^1000")


## End(Not run)
```

print.polyOptim *Pretty printing of polyOptim (Bertini) output.*

Description

Pretty printing of polyOptim (Bertini) output.

Usage

```
## S3 method for class 'polyOptim'
print(x, lagrange = FALSE, digits = 3, ...)
```

Arguments

x	an object of class polyOptim, bertini
lagrange	show values of lagrange multipliers?
digits	digits to round to
...	additional parameters

Value

Invisible string of the printed object.

Examples

```
# see ?polyOptim
```

`print.spectral` *Pretty Printing of Spectral's Output*

Description

Pretty printing of spectral's output.

Usage

```
## S3 method for class 'spectral'
print(x, digits = 3, ...)
```

Arguments

`x`	an object of class spectral
`digits`	digits to round to
`...`	additional parameters

Value

Invisible string of the printed object.

Examples

```
# see ?spectral
```

`print.tableau` *Pretty printing of tableau output.*

Description

Pretty printing of tableau output.

Usage

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

Arguments

`x`	an object of class tableau
`...`	...

Value

Invisible string of the printed object.

Examples

```
# see ?tableau
```

`projectOnto` *Vector Projection onto col(A)*

Description

Project a vector onto the column space of a matrix.

Usage

```
projectOnto(A, x)
```

Arguments

`A`	a matrix
`x`	a vector

Value

...

See Also

`qr.fitted`

Examples

```
A <- diag(5)[,1:2]
x <- 1:5
projectOnto(A, x)
```

`projectOntoPerp` *Vector Projection onto the orthogonal complement of col(A)*

Description

Project a vector onto the orthogonal complement of the column space of a matrix; the null space of A transpose

Usage

```
projectOntoPerp(A, x)
```

Arguments

A	a matrix
x	a vector

Value

...

Examples

```
A <- diag(5)[,1:2]
x <- 1:5
projectOnto(A, x)
```

---

rvotes *Random Spectral Data*

Description

Generate spectral data for testing purposes.

Usage

```
rvotes(nVoters, nObjects, kSelected)
```

Arguments

nVoters	number of voters voting
nObjects	number of objects up for selection
kSelected	number of objects selected by each voter

Value

...

Examples

```
rvotes(100, 10, 3)
```

setBertiniPath *Set Bertini Path*

Description

This function sets the Bertini path either by (1) passing it a character string or (2) using file.choose.

Usage

```
setBertiniPath(path)
```

Arguments

path	a character string, the path to Bertini

Value

invisible bertiniPath

Examples

```
## Not run:

setBertiniPath()


## End(Not run)
```

setLattePath *Set LattE Path*

Description

This function sets the LattE path either by (1) passing it a character string or (2) using file.choose.

Usage

```
setLattePath(path)
```

Arguments

path	a character string, the path to LattE (the function count, for example)

Value

invisible lattePath

Examples

```
## Not run:

setLattePath()


## End(Not run)
```

`setM2Path` *Set Macaulay2 Path*

Description

This function sets the Macaulay2 path either by (1) passing it a character string or (2) using file.choose.

Usage

```
setM2Path(path)
```

Arguments

`path`	a character string, the path to m2

Value

invisible m2Path

Examples

```
## Not run:

setM2Path()


## End(Not run)
```

`setMarkovPath` *Set 4ti2 Path*

Description

This function sets the 4ti2 path either by (1) passing it a character string or (2) using file.choose.

Usage

```
setMarkovPath(path)
```

Arguments

`path`	a character string, the path to 4ti2 (the function markov, for example)

Value

invisible markovPath

Examples

```
## Not run:

setMarkovPath()


## End(Not run)
```

`Smaker` *Means matrix (rank data)*

Description

Compute the means matrix for a full ranking of m objects

Usage

```
Smaker(m)
```

Arguments

`m`	the number of objects

Details

This is the transpose of the means matrix presented in Marden (1995); it projects onto the means subspace of a collection of ranked data. See the examples for how to compute the average rank.

Value

...

References

Marden, J. I. (1995). *Analyzing and Modeling Rank Data*, London: Chapman & Hall. p.41.

See Also

`Tmaker`, `Amaker`, `Emaker`, `Mmaker`, `Pmaker`

Examples

```
data(city)

X <- permutations(3)

# the average rank can be computed without this function
normalize <- function(x) x / sum(x)
factorial(3) * apply(t(X) %*% city, 2, normalize)
# the dataset city is really like three datasets; they can be pooled back
# into one via:
rowSums(city)
factorial(3) * apply(t(X) %*% rowSums(city), 2, normalize)


# the means matrix is used to summarize the data to the means subspace
# which is the subspace of m! spanned by the columns of permutations(m)
# note that when we project onto that subspace, the projection has the
# same average rank vector :
Smaker(3) %*% city # the projections, table 2.8
factorial(3) * apply(t(X) %*% Smaker(3) %*% city, 2, normalize)

# the residuals can be computed by projecting onto the orthogonal complement
(diag(6) - Smaker(3)) %*% city # residuals


apply(t(X) %*% city, 2, function(x) x / sum(x) * factorial(3)) # average ranks by group

apply(t(X) %*% rowSums(city), 2, function(x) x / sum(x) * factorial(3)) # average ranks pooled
```

`spectral` *Analyze a Rank Dataset*

Description

`spectral` analyzes a rank dataset for order interactions; see examples for details.

Usage

```
spectral(data, n, k, levels, iter = 10000, burn = 1000, thin = 10)
```

Arguments

`data`	a vector in the frequency format (see examples)
`n`	the number of objects to select from
`k`	the number of objects selected
`levels`	the names of the outcomes, in the proper order
`iter`	iterations in metropolis
`burn`	burn in
`thin`	thinning

Value

a list containing named elements

- `effects`: the pure ith order effects as a data frame computed by projecting the data onto the isotypic subspaces. the length of each is the same as the data, choose(n, k).
- `effectsNorms`: the l2 norms of each effect.
- `statsMatrices`: the lower order statistics calculating matrices, made with Tmaker.
- `moves`: the markov moves for moving around each V, computed with markov on the statsMatrices. only the positive moves are given.
- `samps`: the samples from each space conditioned on each level of statistics. this is the output of metropolis.
- `obs`: a list of the observed data and its lower level summaries.
- `exp`: a list of the expected number of samples at each level given the summary statistics at the previous (lower) level. these are computed from the samples from metropolis by (1) summarizing them with Tmaker and then (2) averaging the samples. the expected V2 samples (for example), are determined by taking the samples with the same V1 statistics (in V3, say), summarizing them to V2 with Tmaker, and then averaging every cell across the samples.
- `fullExp`: this is the result of taking each of the samples with the same lower-order statistics and averaging them. see exp, which is a reduction of fullExp.
- `residuals`: obs - exp
- `isotypicBases`: a list of the basis vectors of each isotypic subspace; computed as the eigenvalues of the result of Amaker, and grouped by eigenvalue.
- `sampsEffects`: the effects determined by each of the samples, projected onto the isotypic subspaces.
- `sampsEffectsNorms` : the norms of the effects of the samples.
- `sampsEffectsNormSummary` : a summary of the norms of the effects of the samples.
- `showStages`: a function that prints out the observed, expected, and residuals of sequentially conditioning on the sample size, first order statistics, second order statistics, and so on.
- `showFit`: a function that prints out a summary of the fit of the model.
- `decompose`: a function that takes a vector of the same length of the table given and summarizes it to its lower level statistics.
- `sampsDecomposed`: every sample decomposed.
- `statistic`: the pearson's chi-squared (X2), likelihood ratio (G2), Freeman-Tukey (FT), Cressie-Read (CR), and Neyman modified chi-squared (NM) statistics computed using the observed data (obs) and expected data (exp) at each level.
- `sampsStats`: the statistics computed on each of the samples.
- `p.value`: the exact p-values of individual tests, accurate to Monte-Carlo error. these are computed as the proportion of samples with statistics equal to or larger than the oberved statistic.
- `p.value.se`: the standard errors of the p-values computed using the standard asymptotic formula of sqrt(p(1-p)/n). a measure of the Monte-Carlo error.

Examples

```
## Not run:



## voting statistics at different levels
############################################################

# load the cookies dataset:
data(cookie)
cookie$freq
cookie$cookies


# performing the spectral analysis
(out <- spectral(cookie$freq, 6, 3, cookie$cookies))


out$obs # the original observations, and the summary statistics

out$exp # each level is conditional on the previous level's statistics
        # (e.g. what you would expect for 1st order effects given sample size)
        # these are generated using 10k markov bases based mcmc samples

out$p.value # these are approximate exact test p-values using various
            # popular test statistics.  the approximations are good to
            # monte carlo error

out$p.value.se # these are the standard errors using the sqrt(p*(1-p)/n)
               # asymptotic formula, known to have poor performance
               # for small/large p; see package binom for better

out$statistic # the individual statistics are also available
              # the values are not comprable across Vi levels (the rows)
              # as they have asymptotic chi-squared distributions with
              # different degrees of freedom

out$fullExp # you can also get the expected number of samples at each scale
            # for tables with the same ith order statistics, i = 0, ..., k-1


# these can be seen to (re)construct an expected picture of the
# complete data given each successive collection of statistics
cbind(
  obs = cookie$freq,
  as.data.frame(lapply(out$fullExp, function(x) round(x[[4]],1)))
)[c(2:4,1)]
# notice that the reconstruction given only the first order statistics
# (the number of individual cookies selected) is quite good



# instead of using the reconstructions from the exp coming from
# the samples, you could reconstruct the summaries of the observed
# data using bump; it's not quite as good :
V0 <- bump(cookie$freq, 6, 3, 3, 0)
```

```
V1 <- bump(cookie$freq, 6, 3, 3, 1)
V2 <- bump(cookie$freq, 6, 3, 3, 2)

cbind(
  obs = cookie$freq,
  round(data.frame(
    V0 = bump(V0, 6, 3, 0, 3),
    V1 = bump(V1, 6, 3, 1, 3),
    V2 = bump(V2, 6, 3, 2, 3)
  ), 2)
)[c(2:4,1)]



# you can see the model step-by-step with showStages() :
out$showStages()
# notice (1) the significant reduction in the residuals after conditioning
# on the first order statistics and also (2) the powdery noise after
# conditioning on the second order statistics.
# the p-values reflect the same:
#   * the residuals from conditioning on the sample size show the first
#     order effects are strongly significant (in out$p.value V1 = 0)
#   * the residuals from conditioning on the first order effects suggest
#     the second order effects might be significant (V2 ~ .04-.13ish)
#   * the residuals from conditioning on the second order effects indicate
#     the third order effects are entirely insignificant (V3 > .2)


# the isotypic subpaces can be used to determine the pure order effects :

out$isotypicBases # bases of the isotypic subspaces (here 4)

out$effects # pure ith order effects; cookie$freq projected onto the bases
            # these are their effects at the data level, so they all have
            # the same length as the original dataset: choose(n, k)

zapsmall(rowSums(out$effects)) # the effects sum to the data


# if the Vk effects are 0, then the conclusion is that Vk is perfectly
# predicted with the (k-1)st level statistics.  this may lead to the
# conclusion that the l2 norms (say) of the effects might be used to
# gauge the relative strength of effects :
out$effectsNorms # = apply(out$effects, 2, lpnorm)


# the natural (not full-dimensional) residuals can be seen with the summary
out
# or with
out$residuals
# these are the residuals (obs ith level stats) - (exp ith level stats)
# given the (i-1)st statistics
```

```
# bump is a useful function :
out$obs
bump(cookie$freq, 6, 3, 3, 0) # the 0 level is the number of voters, not votes
bump(cookie$freq, 6, 3, 3, 1)
bump(cookie$freq, 6, 3, 3, 2)
bump(cookie$freq, 6, 3, 3, 3)

V1 <- out$obs$V1 # = bump(cookie$freq, 6, 3, 3, 1)
bump(V1, 6, 3, 1, 0)
bump(V1, 6, 3, 1, 1)
bump(V1, 6, 3, 1, 2) # cbind(bump(V1, 6, 3, 1, 2), out$exp$V2)
bump(V1, 6, 3, 1, 3) # cbind(bump(V1, 6, 3, 1, 3), out$fullExp$V1[[4]])
# the differences here are between an observation and an expectation
```

```
out$obs$V1 - out$exp$V1
out$residuals$V1
out$decompose(out$effects$V1)$V1

out$obs$V2 - out$exp$V2
out$residuals$V2

  out$decompose(out$effects$V0)$V2 +
  out$decompose(out$effects$V1)$V2 +
  out$decompose(out$effects$V2)$V2 -
  out$exp$V2
```

```
# this is how to reconstruct the observation given the effects
# the cols of out$effects are the Vk order effects reconstructed
# from the lower level effects
out$obs$V0
zapsmall(
  out$decompose(out$effects$V0)$V0
)

out$obs$V1
zapsmall(
  out$decompose(out$effects$V0)$V1 +
```

```
  out$decompose(out$effects$V1)$V1
)

out$obs$V2
zapsmall(
  out$decompose(out$effects$V0)$V2 +
  out$decompose(out$effects$V1)$V2 +
  out$decompose(out$effects$V2)$V2
)

out$obs$V3
zapsmall(
  out$decompose(out$effects$V0)$V3 +
  out$decompose(out$effects$V1)$V3 +
  out$decompose(out$effects$V2)$V3 +
  out$decompose(out$effects$V3)$V3
)
zapsmall(rowSums(out$effects))

all(cookie$freq == zapsmall(rowSums(out$effects)))



out$effects$V0
out$effects$V0 + out$effects$V1
out$effects$V0 + out$effects$V2
out$effects$V0 + out$effects$V3



str(out$sampsDecomposed)
as.data.frame(lapply(out$sampsDecomposed, function(l) rowMeans(l$V3)))

eff0 <- rowMeans(out$sampsDecomposed$V0$V3)
cbind(eff0, out$effects$V0)

eff1 <- rowMeans(out$sampsDecomposed$V1$V3 - eff0)
cbind(eff1, out$effects$V1)

eff2 <- rowMeans(out$sampsDecomposed$V2$V3 - eff0 - eff1)
cbind(eff2, out$effects$V2)

sum(eff0)
sum(eff1)
sum(eff2)







str(out$sampsEffectsNorms)

data <- out$sampsEffectsNorms$V0$V3
plot(density(data))
curve(dnorm(x, mean(data), sd(data)), col = "red", add = TRUE)
```

```
data <- out$sampsEffectsNorms$V0$V2
plot(density(data))
curve(dnorm(x, mean(data), sd(data)), col = "red", add = TRUE)

data <- out$sampsEffectsNorms$V0$V1
plot(density(data))
curve(dnorm(x, mean(data), sd(data)), col = "red", add = TRUE)


data <- out$sampsEffectsNorms$V1$V3
plot(density(data))
curve(dnorm(x, mean(data), sd(data)), col = "red", add = TRUE)

data <- out$sampsEffectsNorms$V1$V2
plot(density(data))
curve(dnorm(x, mean(data), sd(data)), col = "red", add = TRUE)


data <- out$sampsEffectsNorms$V2$V3
plot(density(data))
curve(dnorm(x, mean(data), sd(data)), col = "red", add = TRUE)




## how to convert data into the right format
############################################################
# this essentially just uses some clever indexing tricks
# to reorder the data in the way you want

data <- cookie$raw       # an example raw, unordered dataset
levels <- cookie$cookies # the order of the objects you want
levsNndcs <- 1:length(levels)
names(levsNndcs) <- levels


# arrange selections within rows (order of selection doesn't matter)
data <- t(apply(data, 1, function(x) x[order(levsNndcs[x])] ))


# arrange rows (order of selectors doesn't matter)
for(k in ncol(data):1) data <- data[order(levsNndcs[data[,k]]),]


# check that you've done the right thing
all( data == cookie$sorted )

# the data frequency order should match that of subsets:
subsets(levels, 1)

subsets(levels, 2)
sapply(subsets(levels, 2), paste, collapse = ", ")

subsets(levels, 3)
```

```
sapply(subsets(levels, 3), paste, collapse = ", ")

names(cookie$freq)
names(cookie$freq) == sapply(subsets(levels, 3), paste, collapse = ", ")









## other examples
############################################################

# rvotes provides uniform samples

n <- 4
k <- 2

raw <- rvotes(250, n, k)
rawTogether <- apply(raw, 1, paste, collapse = " ")
levels <- sapply(subsets(n, k), paste, collapse = " ")
freq <- table( factor(rawTogether, levels = levels) )
(out <- spectral(freq, n, k))

out$p.value
out$showStages()

out$obs
out$exp




n <- 6
k <- 3
raw <- rvotes(250, n, k)
rawTogether <- apply(raw, 1, paste, collapse = " ")
levels <- sapply(subsets(n, k), paste, collapse = " ")
freq <- table( factor(rawTogether, levels = levels) )
(out <- spectral(freq, n, k))


n <- 7
k <- 3
raw <- rvotes(250, n, k)
rawTogether <- apply(raw, 1, paste, collapse = " ")
```

```
levels <- sapply(subsets(n, k), paste, collapse = " ")
freq <- table( factor(rawTogether, levels = levels) )
(out <- spectral(freq, n, k))



n <- 8
k <- 3
raw <- rvotes(250, n, k)
rawTogether <- apply(raw, 1, paste, collapse = " ")
levels <- sapply(subsets(n, k), paste, collapse = " ")
freq <- table( factor(rawTogether, levels = levels) )
# out <- spectral(freq, n, k) # breaks




## End(Not run)
```

`subsets` *Compute Subsets*

Description

Compute the subsets of a given set.

Usage

```
subsets(set, sizes = 1:length(set), include_null = FALSE)
```

Arguments

`set`	the original set
`sizes`	desired size(s) of subsets
`include_null`	should the empty vector be included?

Details

Note that this algorithm is run in R: it is therefore not intended to be the most efficient algorithm for computins subsets.

Value

a list of subsets as vectors

See Also

`combn`

Examples

```
subsets(1:3)
subsets(1:3, size = 2)
subsets(1:3, include_null = TRUE)

subsets(c('a','b','c','d'))
subsets(c('a','b','c','d'), include_null = TRUE)
```

---

summary.bertini *Summarize Bertini Output*

Description

This function summarizes the output from Bertini.

Usage

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

Arguments

object	an object of class bertini
...	additional parameters

Value

Invisible string of the printed object.

Examples

```
# see ?bertini
```

---

tab2vec *Array to Vector conversion*

Description

Convert an array into a vector.

Usage

```
tab2vec(tab)
```

Arguments

tab	an array of counts

Details

This function converts an array (or a multi-way contingency table) into a vector, using a consistent ordering of the cells. The ordering of the cells is lexicographical and cannot be specified by the user.

Value

a named integer vector. the names correspond to the cell indices in the table.

See Also

vec2tab

Examples

```
a <- array(1:24, c(2,3,4))
tab2vec(a)

data(Titanic)
tab2vec(Titanic)
Titanic[1,1,1,1]
Titanic[1,1,1,2]
```

---

tableau *Tableau Notation for Markov*

Description

Print the tableau notation for a Markov move. See the reference provided, p. 13.

Usage

```
tableau(move, dim)
```

Arguments

move	a markov move matrix, where the columns are moves in vector form (e.g. the output of markov)
dim	the dimensions of the table form of the move, oftentimes a vector of the number of levels of each variable in order

Value

an object of class tableau

References

Drton, M., B. Sturmfels, and S. Sullivant (2009). *Lectures on Algebraic Statistics*, Basel: Birkhauser Verlag AG.

Examples

```
## Not run:

# 2x2 independence example
# following convention, the first index indicates rows
varlvls <- c(2,2)
facets <- list(1,2)
( A <- hmat(varlvls, facets) )
markov(A)
markov(A, "vec")
markov(A, "tab", varlvls)
markov(A, "tab", varlvls, TRUE)
tableau(markov(A), varlvls)







# LAS example 1.2.12, p.17  (no 3-way interaction)
varlvls <- c(2,2,2)
facets <- list(c(1,2), c(1,3), c(2,3))
( A <- hmat(varlvls, facets) )
markov(A)







## End(Not run)
```

`teshape` *Interconvert data structures*

Description

Interconvert an array, a raw data frame, and frequency distribution data.frame.

Usage

```
teshape(data, out = c("freq", "tab", "raw"))
```

Arguments

`data`	a data frame or array
`out`	the output format, see examples

Details

Multivariate categorical data can be represented in several ways. Three comon ways are : a contingency table, a data frame of raw observations (1 row = 1 subject), and a long data frame with a variable containing the counts in the contingency table.

Value

a matrix containing the Markov basis as its columns (for easy addition to tables)

Examples

```
data(Titanic)

# array to others
teshape(Titanic, "freq")
teshape(Titanic, "tab") # what it was
teshape(Titanic, "raw")


# freq to others
TitanicFreq <- teshape(Titanic, "freq")
teshape(TitanicFreq, "freq") # what it was
teshape(TitanicFreq, "tab")  # == Titanic
teshape(TitanicFreq, "raw")

# raw to others
TitanicRaw <- teshape(Titanic, "raw")
teshape(TitanicRaw, "freq")
teshape(TitanicRaw, "tab")
teshape(TitanicRaw, "raw")
```

Tmaker *Create the sufficient statistics calculating matrix for approval data*

Description

Create the sufficient statistics calculating matrix for approval data

Usage

```
Tmaker(m, k, d)
```

Arguments

m	the number of objects
k	the number of objects selected
d	the order-effect for the desired matrix (0 to k)

Value

...

See Also

Emaker, Amaker, Mmaker, Pmaker, Smaker

Examples

```
Tmaker(4, 2, 0) # m
Tmaker(4, 2, 1) # generates how many of each
Tmaker(4, 2, 2) # gives data (order = subsets(1:4, 2))

Tmaker(5, 2, 0)
Tmaker(5, 2, 1)
Tmaker(5, 2, 2)

Tmaker(4, 3, 0) #
Tmaker(4, 3, 1) # subsets(1:4, 3), 1 is in 1, 2, and 3
Tmaker(4, 3, 2) # subsets(1:4, 2)
Tmaker(4, 3, 3)
```

```
data(cookie)


## voting statistics at different levels
############################################################

# projection onto V0: the number of people in survey
effectsOnV0 <- Tmaker(6, 3, 0) %*% cookie$freq
colnames(effectsOnV0) <- "Total Votes"
effectsOnV0 # = sum(cookie$freq)


# projection onto V1: the number of people voting for each cookie
effectsOnV1 <- Tmaker(6, 3, 1) %*% cookie$freq
row.names(effectsOnV1) <- cookie$cookies
colnames(effectsOnV1) <- "Total Votes"
effectsOnV1


# projection onto V2: the number of people voting for each cookie-pair
effectsOnV2 <- Tmaker(6, 3, 2) %*% cookie$freq
row.names(effectsOnV2) <- sapply(subsets(cookie$cookies, 2), paste, collapse = ", ")
colnames(effectsOnV2) <- "Total Votes"
effectsOnV2


# projection onto V3: the number of people voting for each cookie-triple
effectsOnV3 <- Tmaker(6, 3, 3) %*% cookie$freq
row.names(effectsOnV3) <- sapply(subsets(cookie$cookies, 3), paste, collapse = ", ")
colnames(effectsOnV3) <- "Total Votes"
effectsOnV3 # = t(t(cookie$freq)) = the (freq) data
```

Umaker *U matrix (rank data)*

Description

Compute the generalized marginals matrix for a full ranking of m objects. Umaker generalized Mmaker.

Usage

```
Umaker(m)
```

Arguments

m	the number of objects

Details

This is the transpose of the generalized marginals matrix presented in Marden (1995).

Value

...

References

Marden, J. I. (1995). *Analyzing and Modeling Rank Data*, London: Chapman & Hall. pp.47–48.

See Also

Mmaker, Pmaker, Smaker

Examples

```
data(politicalGoals)

lambdas <- apply(partitions(4), 1, function(v) v[v != 0])
```

upper *Create an upper triangular matrix*

Description

Create an upper triangular matrix.

Usage

```
upper(x)
```

Arguments

x	a vector

Value

...

See Also

lower

Examples

```
upper(1:3)
lower(1:3)

upper(1:6)
lower(1:6)

upper(rnorm(6))
```

---

variety *Compute a Variety*

Description

The variety of a collection of multivariate polynomials is the collection of points at which those polynomials are (simultaneously) equal to 0. `variety` uses Bertini to find this set.

Usage

```
variety(mpolyList, varOrder, ...)
```

Arguments

mpolyList	Bertini code as either a character string or function; see examples
varOrder	variable order (see examples)
...	stuff to pass to bertini

Value

an object of class bertini

Examples

```
## Not run:


polys <- mp(c(
  "x^2 - y^2 - z^2 - .5",
  "x^2 + y^2 + z^2 - 9",
  ".25 x^2 + .25 y^2 - z^2"
```

```
))
variety(polys)

# algebraic solution :
c(sqrt(19)/2, 7/(2*sqrt(5)), 3/sqrt(5)) # +/- each ordinate



# character vectors can be taken in; they're passed to mp
variety(c("y - x^2", "y - x - 2"))



# an example of how varieties are invariant to the
# the generators of the ideal
variety(c("2 x^2 + 3 y^2 - 11", "x^2 - y^2 - 3"))

# the following takes a few seconds to initialize, feel free to them
# gb <- grobner(mp(c("2 x^2 + 3 y^2 - 11", "x^2 - y^2 - 3")))
# variety(gb)

m2("
R = QQ[x,y]
gens gb ideal(2*x^2 + 3*y^2 - 11, x^2 - y^2 - 3)
")
variety(c("y^2 - 1", "x^2 - 4"))
variety(c("x^2 - 4", "y^2 - 1"))



# variable order is by default equal to vars(mpolyList)
# (this finds the zeros of y = x^2 - 1)
variety(c("y", "y - x^2 + 1")) # y, x
vars(mp(c("y", "y - x^2 + 1")))
variety(c("y", "y - x^2 + 1"), c("x", "y")) # x, y



# complex solutions
variety("x^2 + 1")
variety(c("x^2 + 1 + y", "y"))


# multiplicities
variety("x^2")
variety(c("2 x^2 + 1 + y", "y + 1"))
variety(c("x^3 - x^2 y", "y + 2"))


#
p <- mp(c("2 x  -  2  -  3 x^2 l  -  2 x l",
  "2 y  -  2  +  2 l y",
  "y^2  -  x^3  -  x^2"))
variety(p)


## End(Not run)
```

vec2tab *Vector to Array conversion*

Description

Convert a vector into an array given a set of dimensions; it therefore simply wraps `aperm` and `array`.

Usage

```
vec2tab(vec, dim)
```

Arguments

`vec`	a vector
`dim`	the desired array dimensions, oftentimes a vector of the number of levels of each variable in order

Details

This function converts an array (or a multi-way contingency table) into a vector, using a consistent ordering of the cells. The ordering of the cells is lexicographical and cannot be specified by the user.

Value

an array

See Also

`tab2vec`, `aperm`, `array`

Examples

```
data(Titanic)
Titanic
tab2vec(Titanic)
vec2tab(tab2vec(Titanic), dim(Titanic))
vec2tab(tab2vec(Titanic), dim(Titanic)) == Titanic
all(vec2tab(tab2vec(Titanic), dim(Titanic)) == Titanic)
```

Index