Package: latte (via r-universe)

September 2, 2024

Type Package

Title Interface to 'LattE' and '4ti2'

Version 0.2.1.900

Maintainer David Kahle <david@kahle.io>

Description Back-end connections to 'LattE' (<https://www.math.ucdavis.edu/~latte>) for counting lattice points and integration inside convex polytopes and '4ti2' (<http://www.4ti2.de/>) for algebraic, geometric, and combinatorial problems on linear spaces and front-end tools facilitating their use in the 'R' ecosystem.

License GPL-2

URL https://github.com/dkahle/latte

BugReports https://github.com/dkahle/latte/issues

LazyData TRUE

SystemRequirements LattE <https://www.math.ucdavis.edu/~latte/>, 4ti2 <http://www.4ti2.de/>

Imports magrittr, stringr, mpoly, ggplot2, memoise, dplyr, usethis, glue

Suggests knitr, rmarkdown

RoxygenNote 7.2.0

Roxygen list(markdown = TRUE)

Encoding UTF-8

Repository https://dkahle.r-universe.dev

RemoteUrl https://github.com/dkahle/latte

RemoteRef HEAD

RemoteSha 927b37ceb755bc897ad936af4e8812f54a85a252

Contents

`call_latte` *Unified interface to calling LattE*

Description

Unified interface to calling LattE

Usage

```
call_latte(
  which_exe,
  input_file,
  project_dir,
  opts = list(),
  stdout = "stdout",
  stderr = "stderr",
  shell = FALSE
)
```

Arguments

`which_exe`	The name of LattE executable, e.g. `"count"`
`input_file`	The filename of file to be given to the executable, without a path
`project_dir`	The path associated with `input_file`.

opts	A named list of optional command line arguments to the executable.
stdout, stderr	Filenames to redirect standard output and standard error to.
shell	logical(1); should the shell command be messaged to the user?

Value

invisible()

---

genmodel *Generate a configuration matrix*

Description

genmodel runs 4ti2's genmodel program to compute the configuration matrix A corresponding to graphical statistical models given by a simplicial complex and levels on the nodes.

Usage

```
genmodel(varlvls, facets, dir = tempdir(), quiet = TRUE, shell = FALSE, ...)
```

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
dir	Directory to place the files in, without an ending /
quiet	If FALSE, messages the 4ti2 output
shell	Messages the shell code used to do the computation
...	Additional arguments to pass to the function

Value

The configuration matrix of the model provided

Examples

```
if (has_4ti2()) {

varlvls <- rep(2, 2)
facets <- list(1, 2)
genmodel(varlvls, facets)
genmodel(varlvls, facets, quiet = FALSE)

varlvls <- rep(3, 3)
facets <- list(1:2, 2:3, c(3,1))
genmodel(varlvls, facets)

# compare this to algstat's hmat function
```

```
}
```

---

kprod *Iterated Kronecker product*

Description

Compute the Kronecker product of several matrices.

Usage

```
kprod(..., FUN = `*`)
```

Arguments

...	A listing of matrices
FUN	A function to pass to kronecker()

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 the specified 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))


# cf. aoki, hara, and takemura p.13
rbind(
  kprod(diag(2), t(ones(2))),
  kprod(t(ones(2)), diag(2))
)
```

latte *R Interface to LattE and 4ti2*

Description

Back-end connections to LattE (https://www.math.ucdavis.edu/~latte/) and 4ti2 (http://www.4ti2.de/) executables and front-end tools facilitating their use in the R ecosystem.

latte-count *Count integer points in a polytope*

Description

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

Usage

```
count_core(spec, dir = tempdir(), quiet = TRUE, mpoly = TRUE, ...)

latte_count(spec, dir = tempdir(), quiet = TRUE, mpoly = TRUE, ...)

latte_fcount(spec, dir = tempdir(), quiet = TRUE, mpoly = TRUE, ...)
```

Arguments

`spec`	Specification, see details and examples
`dir`	Directory to place the files in, without an ending /
`quiet`	Show latte output?
`mpoly`	When opts = "–ehrhart-polynomial", return the mpoly version of it
`...`	Additional arguments to pass to the function, see count –help at the command line to see examples. Note that dashes - should be specified with underscores _

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, (3) a list of A and b for the equation Ax <= b (see examples), or (4) raw code for LattE's count program. If a character vector is supplied, (1) and (4) 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

```
if (has_latte()) {

spec <- c("x + y <= 10", "x >= 1", "y >= 1")
latte_count(spec) # 45
latte_count(spec, quiet = FALSE) # 45
latte_count(spec, dilation = 10) # 3321
latte_count(spec, homog = TRUE) # 45

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

# ehrhart polynomials
latte_count(spec, ehrhart_polynomial = TRUE)
latte_count(spec, ehrhart_polynomial = TRUE, mpoly = FALSE)

# ehrhart series (raw since mpoly can't handle rational functions)
latte_count(spec, ehrhart_series = TRUE)

# simplified ehrhart series - not yet implemented
#latte_count(spec, simplified_ehrhart_polynomial = TRUE)

# first terms of the ehrhart series
latte_count(spec, ehrhart_taylor = 1)
# latte_count(spec, ehrhart_taylor = 2)
# latte_count(spec, ehrhart_taylor = 3)
# latte_count(spec, ehrhart_taylor = 4)

# multivariate generating function
latte_count(spec, multivariate_generating_function = TRUE)


# by vertices
spec <- list(c(1,1), c(10,1), c(1,10), c(10,10))
latte_count(spec)
latte_count(spec, vrep = TRUE)

code <- "
5 3
1 -1  0
1  0 -1
1 -1 -1
0  1  0
0  0  1
"
```

```
latte_count(code)


# for Ax <= b, see this example from the latte manual p.10
A <- matrix(c(
   1,  0,
   0,  1,
   1,  1,
  -1,  0,
   0, -1
), nrow = 5, byrow = TRUE)
b <- c(1, 1, 1, 0, 0)
latte_count(list(A = A, b = b))









}
```

`latte-files` *Format/read/write a matrix in latte's style*

Description

`format_latte()` formats a matrix in latte's style. `write_latte()` writes a latte-formatted file to file. `read_latte()` reads a latte-formatted file from disk.

Usage

```
format_latte(mat, file)

write_latte(mat, file)

write.latte(mat, file)

read_latte(file, format = c("mat", "Ab"))

read.latte(file, format = c("mat", "Ab"))
```

Arguments

`mat`	A matrix

file	A filename
format	"mat" or "Ab"

Value

- format_latte() – A character string of the matrix in latte format.
- write_latte() – An invisible character string of the formatted output.
- read_latte() – An integer matrix.

Examples

```
(mat <- matrix(sample(9), 3, 3))

format_latte(mat)
cat(format_latte(mat))

(file <- file.path(tempdir(), "foo.hrep"))
write_latte(mat, file)
file.show(file)
read_latte(file)
read_latte(file, "Ab")

attr(mat, "linearity") <- c(1, 3)
attr(mat, "nonnegative") <- 2
mat
format_latte(mat)
cat(format_latte(mat))
write_latte(mat, file)
file.show(file)
read_latte(file)

file.remove(file)
```

---

latte-optim *Solve an integer progam with LattE*

Description

latte_max and latte_min use LattE's latte-maximize and latte-minimize functions to find the maximum or minimum of a linear objective function over the integers points in a polytope (i.e. satisfying linearity constraints). This makes use of the digging algorithm; see the LattE manual at http://www.math.ucdavis.edu/~latte for details.

Usage

```
latte_optim(
  objective,
  constraints,
  type = c("max", "min"),
  method = c("lp", "cones"),
  dir = tempdir(),
  opts = "",
  quiet = TRUE,
  shell = FALSE
)

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

latte_min(
  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
`type`	`"max"` or `"min"`
`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
`shell`	Messages the shell code used to do the computation

Value

A named list with components `par`, a named-vector of optimizing arguments, and `value`, the value of the objective function at the optimial point.

Examples

```
if (has_latte()) {

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

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


df <- expand.grid("x" = 0:10, "y" = 0:10)
df <- subset(df, x + y <= 10L)
df$objective <- with(df, -2*x + 3*y)
library("ggplot2")
ggplot(df, aes(x, y, size = objective)) +
  geom_point()

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



latte_min("-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)




}
```

`lattice-bases` *Compute a basis with 4ti2*

Description

4ti2 provides several executables that can be used to generate bases for a configuration matrix A. See the references for details.

Usage

```
basis(exec, memoise = TRUE)

zbasis(
  A,
  format = c("mat", "vec", "tab"),
  dim = NULL,
  all = FALSE,
  dir = tempdir(),
  quiet = TRUE,
  shell = FALSE,
  dbName = NULL,
  ...
)

markov(
  A,
  format = c("mat", "vec", "tab"),
  dim = NULL,
  all = FALSE,
  dir = tempdir(),
  quiet = TRUE,
  shell = FALSE,
  dbName = NULL,
  ...
)

groebner(
  A,
  format = c("mat", "vec", "tab"),
  dim = NULL,
  all = FALSE,
  dir = tempdir(),
  quiet = TRUE,
  shell = FALSE,
  dbName = NULL,
  ...
)

hilbert(
  A,
  format = c("mat", "vec", "tab"),
  dim = NULL,
  all = FALSE,
  dir = tempdir(),
  quiet = TRUE,
  shell = FALSE,
  dbName = NULL,
```

```
  ...
)

graver(
  A,
  format = c("mat", "vec", "tab"),
  dim = NULL,
  all = FALSE,
  dir = tempdir(),
  quiet = TRUE,
  shell = FALSE,
  dbName = NULL,
  ...
)

fzbasis(
  A,
  format = c("mat", "vec", "tab"),
  dim = NULL,
  all = FALSE,
  dir = tempdir(),
  quiet = TRUE,
  shell = FALSE,
  dbName = NULL,
  ...
)

fmarkov(
  A,
  format = c("mat", "vec", "tab"),
  dim = NULL,
  all = FALSE,
  dir = tempdir(),
  quiet = TRUE,
  shell = FALSE,
  dbName = NULL,
  ...
)

fgroebner(
  A,
  format = c("mat", "vec", "tab"),
  dim = NULL,
  all = FALSE,
  dir = tempdir(),
  quiet = TRUE,
  shell = FALSE,
  dbName = NULL,
```

```
  ...
)

fhilbert(
  A,
  format = c("mat", "vec", "tab"),
  dim = NULL,
  all = FALSE,
  dir = tempdir(),
  quiet = TRUE,
  shell = FALSE,
  dbName = NULL,
  ...
)

fgraver(
  A,
  format = c("mat", "vec", "tab"),
  dim = NULL,
  all = FALSE,
  dir = tempdir(),
  quiet = TRUE,
  shell = FALSE,
  dbName = NULL,
  ...
)
```

Arguments

`exec`	don't use this parameter
`memoise`	don't use this parameter
`A`	The configuration matrix
`format`	How the basis (moves) should be returned. if "mat", the moves are returned as the columns of a matrix.
`dim`	The dimension to be passed to `vec2tab()` if format = "tab" is used; 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 as returned by the executable.
`dir`	Directory to place the files in, without an ending /
`quiet`	If FALSE, messages the 4ti2 output
`shell`	Messages the shell code used to do the computation
`dbName`	The name of the model in the markov bases database, http://markov-bases.de, see examples
`...`	Additional arguments to pass to the function, e.g. `p = "arb"` specifies the flag `-parb`; not setting this issues a common warning

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

```
if (has_4ti2()) {


# basic input and output for the 3x3 independence example
(A <- rbind(
  kprod(diag(3), ones(1,3)),
  kprod(ones(1,3), diag(3))
))
markov(A, p = "arb")



# you can get the output formatted in different ways:
markov(A, p = "arb", all = TRUE)
markov(A, p = "arb", "vec")
markov(A, p = "arb", "tab", c(3, 3))
tableau(markov(A, p = "arb"), dim = c(3, 3)) # tableau notation



# you can add options by listing them off
# to see the options available to you by function,
# go to http://www.4ti2.de
markov(A, p = "arb")



# the basis functions are automatically cached for future use.
# (note that it doesn't persist across sessions.)
A <- rbind(
  kprod(  diag(4), ones(1,4), ones(1,4)),
  kprod(ones(1,4),   diag(4), ones(1,4)),
  kprod(ones(1,4), ones(1,4),   diag(4))
)
system.time(markov(A, p = "arb"))
system.time(markov(A, p = "arb"))

# the un-cashed versions begin with an "f"
# (think: "forgetful" markov)
system.time(fmarkov(A, p = "arb"))
system.time(fmarkov(A, p = "arb"))
```

```
# you can see the command line code by typing shell = TRUE
# and the standard output wiht quiet = FALSE
# we illustrate these with fmarkov because otherwise it's cached
(A <- rbind(
  kprod(diag(2), ones(1,4)),
  kprod(ones(1,4), diag(2))
))
fmarkov(A, p = "arb", shell = TRUE)
fmarkov(A, p = "arb", quiet = FALSE)



# compare the bases for the 3x3x3 no-three-way interaction model
A <- rbind(
  kprod(  diag(3),   diag(3), ones(1,3)),
  kprod(  diag(3), ones(1,3),   diag(3)),
  kprod(ones(1,3),   diag(3),   diag(3))
)
str(  zbasis(A, p = "arb")) #    8 elements = ncol(A) - qr(A)$rank
str(  markov(A, p = "arb")) #   81 elements
str(groebner(A, p = "arb")) #  110 elements
str(  graver(A))            #  795 elements


# the other bases are also cached
A <- rbind(
  kprod(  diag(3), ones(1,3), ones(1,2)),
  kprod(ones(1,3),   diag(3), ones(1,2)),
  kprod(ones(1,3), ones(1,3),   diag(2))
)
system.time( graver(A))
system.time( graver(A))
system.time(fgraver(A))
system.time(fgraver(A))



# LAS ex 1.2.1, p.12 : 2x3 independence
(A <- rbind(
  kprod(diag(2), ones(1,3)),
  kprod(ones(1,2), diag(3))
))

markov(A, p = "arb", "tab", c(3, 3))
# Prop 1.2.2 says that there should be
2*choose(2, 2)*choose(3,2) # = 6
# moves (up to +-1)
markov(A, p = "arb", "tab", c(3, 3), TRUE)
```

```
# LAS example 1.2.12, p.17  (no 3-way interaction)
(A <- rbind(
  kprod(  diag(2),   diag(2), ones(1,2)),
  kprod(  diag(2), ones(1,2),   diag(2)),
  kprod(ones(1,2),   diag(2),   diag(2))
))
plot_matrix(A)
markov(A, p = "arb")
groebner(A, p = "arb")
graver(A)
tableau(markov(A, p = "arb"), dim = c(2,2,2))




# using the markov bases database, must be connected to internet
# commented out for predictable and fast cran checks time
# A <- markov(dbName = "ind3-3")
# B <- markov(rbind(
#   kprod(diag(3), ones(1,3)),
#   kprod(ones(1,3), diag(3))
# ), p = "arb")
# all(A == B)




# possible issues
# markov(diag(1, 10))
# zbasis(diag(1, 10), "vec")
# groebner(diag(1, 10), "vec", all = TRUE)
# graver(diag(1, 10), "vec", all = TRUE)
# graver(diag(1, 4), "tab", all = TRUE, dim = c(2,2))



}
```

---

ones *Ones*

Description

Make an array of ones

Usage

```
ones(...)
```

Arguments

...	A sequence of dimensions separated by commas

Value

An integer array of ones

Examples

```
ones(5)
ones(5, 1)
ones(1, 5)
ones(2, 3)
ones(2, 3, 2)

str(ones(5))
```

---

pathing *Set paths to LattE and 4ti2 executables*

Description

These are helper functions that deal with pathing to external programs and asking if they are present. When latte is loaded it attempts to find LattE and 4ti2 executables by looking for environment variables specifying them, i.e. their paths as specified in your .Renviron file.

Usage

```
set_latte_path(path)

set_4ti2_path(path)

get_4ti2_path()

get_latte_path()

has_4ti2()

has_latte()

missing_4ti2_stop()

missing_latte_stop()
```

Arguments

path	A character string, the path to a 4ti2 function (e.g. markov) for setting 4ti2's path or a LattE function (e.g. count) for LattE's path

Details

For easiest use, you'll want to specify the paths of LattE and 4ti2 executables in your ~/.Renviron file. They should look something like

```
LATTE=/Applications/latte/bin
```

```
FOURTITWO=/Applications/latte/bin
```

You can set these permanently with `edit_r_environ()`.

You can change these for the current session using, e.g., `set_latte_path()`, which accepts a character string or, if missing, uses `file.choose()` to let you interactively; you just select an arbitrary executable.

Value

A logical(1) or character(1) containing the path.

Author(s)

David Kahle <david@kahle.io>

Examples

```
has_4ti2()
if (has_4ti2()) get_4ti2_path()

has_latte()
if (has_4ti2()) get_latte_path()


# you can set these paths permanently with the following. note that you'll
# need to re-start the R session afterwards or simply pass the path into,
# e.g., set_latte_path(). see below for more details on that.
if (interactive()) edit_r_environ()


# you can change these in your current session with set_latte_path() and
if (had_latte <- has_latte()) old_latte_path <- get_latte_path()
set_latte_path("/path/to/latte")
get_latte_path()

if (had_latte) set_latte_path(old_latte_path)
get_latte_path()
```

plot-matrix *Plot a matrix*

Description

plot_matrix is a R variant of Matlab's spy function.

Usage

```
plot_matrix(A)
```

Arguments

A	A matrix

Value

a ggplot object

Author(s)

David Kahle <david@kahle.io>

Examples

```
# the no-three-way interaction configuration
(A <- kprod(ones(1,3), diag(3), ones(3)))
plot_matrix(A)


if (has_4ti2()) {

plot_matrix(markov(A))

(A <- genmodel(c(2L, 2L), list(1L, 2L)))
plot_matrix(A)
plot_matrix(markov(A))

(A <- genmodel(c(5L, 5L), list(1L, 2L)))
plot_matrix(A)
plot_matrix(markov(A))

}
```

ppi *Compute the primitive partition identities*

Description

ppi runs 4ti2's ppi program to compute the primitive partition identities, that is, the Graver basis of 1:N.

Usage

```
ppi(N, dir = tempdir(), quiet = TRUE, shell = FALSE, ...)
```

Arguments

N	A postive integer > 2
dir	Directory to place the files in, without an ending /
quiet	If FALSE, messages the 4ti2 output
shell	Messages the shell code used to do the computation
...	Additional arguments to pass to the function

Value

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

See Also

graver()

Examples

```
if (has_4ti2()) {

ppi(3)
t(ppi(3)) %*% 1:3
plot_matrix(ppi(3))

graver(t(1:3))
plot_matrix(graver(t(1:3)))

ppi(5, quiet = FALSE, shell = TRUE)

}
```

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
```

qsolve *Solve a linear system over the rationals*

Description

qsolve runs 4ti2's qsolve program to compute the configuration matrix A corresponding to graphical statistical models given by a simplicial complex and levels on the nodes.

Usage

```
qsolve(mat, rel, sign, dir = tempdir(), quiet = TRUE, shell = FALSE, ...)
```

Arguments

mat	The A matrix (see the 4ti2 documentation or examples)
rel	A vector of "<" or ">" relations
sign	The signs of the individual
dir	Directory to place the files in, without an ending /
quiet	If FALSE, messages the 4ti2 output
shell	Messages the shell code used to do the computation
...	Additional arguments to pass to the function

Value

The configuration matrix of the model provided

Examples

```
if (has_4ti2()) {

# x + y > 0
# x + y < 0

mat <- rbind(
  c( 1,  1),
  c( 1,  1)
)
rel <- c(">", "<")
sign <- c(0, 0)

qsolve(mat, rel, sign, p = "arb")
qsolve(mat, rel, sign, p = "arb", quiet = FALSE)
qsolve(mat, rel, sign, p = "arb", shell = TRUE)

}
```

---

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(1:6, c(1,2,3))
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

```
vec <- matrix(c(1, -1, -1, 1), nrow = 4)
varlvls <- c(2, 2)
tableau(vec, varlvls)
```

---

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 lexicographic and cannot be specified by the user.

Value

An array

See Also

`tab2vec()`, `aperm()`, `array()`

Examples

```
data(Titanic)
str( Titanic )
str( tab2vec(Titanic) )

# convert it back into a table (names are removed)
vec2tab(
  tab2vec(Titanic),
  dim(Titanic)
)

# check that they are the same
all( vec2tab(tab2vec(Titanic), dim(Titanic)) == Titanic )
```

---

zsolve *Solve a linear system over the integers*

Description

zsolve runs 4ti2's zsolve program to compute the configuration matrix A corresponding to graphical statistical models given by a simplicial complex and levels on the nodes.

Usage

```
zsolve(
  mat,
  rel,
  rhs,
  sign,
  lat,
  lb,
  ub,
  dir = tempdir(),
  quiet = TRUE,
  shell = FALSE,
  ...
)
```

Arguments

mat	The A matrix (see the 4ti2 documentation or examples)
rel	A vector of "<" or ">" relations
rhs	The right hand side b
sign	The signs of the individual
lat	A lattice basis (instead of a matrix)

lb	Lower bounds on columns
ub	Upper bounds on columns
dir	Directory to place the files in, without an ending /
quiet	If FALSE, messages the 4ti2 output
shell	Messages the shell code used to do the computation
...	Additional arguments to pass to the function

Value

The configuration matrix of the model provided

Examples

```
if (has_4ti2()) {

mat <- rbind(
  c( 1, -1),
  c(-3,  1),
  c( 1,  1)
)
rel <- c("<", "<", ">")
rhs <- c(2, 1, 1)
sign <- c(0, 1)

zsolve(mat, rel, rhs, sign)
zsolve(mat, rel, rhs, sign, quiet = FALSE)
zsolve(mat, rel, rhs, sign, shell = TRUE)

zsolve(mat, rel, rhs, sign, p = "gmp", quiet = FALSE)

}
```

Index