Package 'kstMatrix'

November 9, 2026

Type Package

Date 2026-11-09

Version 3.0-1

Title Basic Functions in Knowledge Space Theory Using Matrix Representation

Description Knowledge space theory by Doignon and Falmagne (1999) <doi:10.1007/978-3-642-58625-5> is a set- and order-theoretical framework, which proposes mathematical formalisms to operationalize knowledge structures in a particular domain. The 'kstMatrix' package provides basic functionalities to generate, handle, and manipulate knowledge structures and knowledge spaces. Opposed to the 'kst' package, 'kstMatrix' uses matrix representations for knowledge structures. Furthermore, 'kstMatrix' contains several knowledge spaces obtained in the 1990s by the research group around Cornelia Dowling through querying experts.

Depends R (>= 4.5.0)

Imports stats, grDevices, sets, pks, tidyr, DiagrammeR, rsvg

Suggests DiagrammeRsvg, knitr, markdown, Rgraphviz

Maintainer Cord Hockemeyer <cord.hockemeyer@uni-graz.at>

License GPL-3

NeedsCompilation yes

Repository CRAN

Encoding UTF-8

LazyData true

VignetteBuilder knitr

Config/roxygen2/version 8.1.1

Author Cord Hockemeyer [aut, cre],
Peter Steiner [aut],
Wai Wong [aut]

Date/Publication 2026-11-09 05:10:02 UTC

Contents

binary_matrix_product . . . 3
cad . . . 4
fractions . . . 4
kmassess . . . 5
kmassessbayesian . . . 8
kmassesshalfsplit . . . 9
kmassessinformative . . . 10
kmassessmentsimulation . . . 11
kmassessmultiplicative . . . 12
kmbasis . . . 13
kmbasisdiagram . . . 14
kmbasisfringe . . . 15
kmbasisneighbourhood . . . 16
kmclosure . . . 16
kmcolors . . . 17
kmdist . . . 18
kmdoubleequal . . . 19
kmeqreduction . . . 19
kmexpand . . . 20
kmfamset . . . 21
kmfringe . . . 21
kmgenerate . . . 22
kmgradations . . . 23
kmhasse . . . 24
kmheights . . . 24
kmiita2SR . . . 25
kmintersection . . . 26
kmintersectionclosure . . . 27
kmiswellgraded . . . 27
kmlearningpathmatrices . . . 28
kmlearningpaths . . . 29
kmmesh . . . 30
kmminimalfamset . . . 30
kmneighbourhood . . . 31
kmnotions . . . 32
kmprettyprint . . . 33
kmqspace . . . 36
kmrefine . . . 37
kmsetiselement . . . 38
kmSF2basis . . . 39
kmsimulate . . . 39
kmspace . . . 40
kmSR2basis . . . 41
kmSRdiagram . . . 42
kmSRvalidate . . . 42
kmstructure . . . 43

kmsubstructure 44
kmsurmisefunction 45
kmsurmiserelation 46
kmsymmsetdiff 46
kmtrivial 47
kmunion 48
kmunionclosure 49
kmvalidate 50
phsg 51
plot 51
readwrite 55
xpl 55

Index **57**

`binary_matrix_product` *Compute a binary matrix product*

Description

`binary_matrix_product` expects two binary matrices and computes their Boolean product.

Usage

```
binary_matrix_product(m, n)
```

Arguments

`m`	Binary matrix
`n`	Binary matrix

Value

Boolean matrix product of m and n

See Also

Other Utilities: `kmdoubleequal()`, `kmminimalfamset()`, `kmsetiselement()`, `kmsymmsetdiff()`, `kmtrivial`

Examples

```
binary_matrix_product(xpl$sr, kmsurmiserelation(kmmaximalspace(4)))
binary_matrix_product(xpl$sr, kmsurmiserelation(kmminimalspace(4)))
```

cad *Knowledge spaces on AutoCAD knowledge*

Description

Bases of knowledge spaces on AutoCAD knowledge obtained from querying experts.

Usage

```
cad
```

Format

A list containing seven bases (cad1 to cad6, and cadmaj) in binary matrix form. Each matrix has 28 columns representing the different knowledge items and a varying number of rows containing the basis elements.

Details

Six experts were queried about prerequisite relationships between 28 AutoCAD knowledge items (Dowling, 1991; 1993). A seventh basis represents those prerequisite relationships on which the majority (4 out of 6) of the experts agree (Dowling & Hockemeyer, 1998).

References

Dowling, C. E. (1991). *Constructing Knowledge Structures from the Judgements of Experts.* Habilitationsschrift, Technische Universität Carolo-Wilhelmina, Braunschweig, Germany.

Dowling, C. E. (1993). Applying the basis of a knowledge space for controlling the questioning of an expert. *Journal of Mathematical Psychology, 37,* 21–48.

Dowling, C. E. & Hockemeyer, C. (1998). Computing the intersection of knowledge spaces using only their basis. In Cornelia E. Dowling, Fred S. Roberts, & Peter Theuns, editors, *Recent Progress in Mathematical Psychology,* pp. 133–141. Lawrence Erlbaum Associates Ltd., Mahwah, NJ.

fractions *Knowledge spaces on fractions*

Description

Bases of knowledge spaces on fractions obtained from querying experts.

Usage

```
fractions
```

Format

A list containing four bases (frac1 to frac3, and fracmaj) in binary matrix form. Each matrix has 77 columns representing the different knowledge items and a varying number of rows containing the basis elements.

Details

Three experts were queried about prerequisite relationships between 77 items on fractions (Baumunk & Dowling, 1997). A forth basis represents those prerequisite relationships on which the majority of the experts agree (Dowling & Hockemeyer, 1998).

References

Baumunk, K. & Dowling, C. E. (1997). Validity of spaces for assessing knowledge about fractions. *Journal of Mathematical Psychology, 41,* 99–105.

Dowling, C. E. & Hockemeyer, C. (1998). Computing the intersection of knowledge spaces using only their basis. In Cornelia E. Dowling, Fred S. Roberts, & Peter Theuns, editors, *Recent Progress in Mathematical Psychology,* pp. 133–141. Lawrence Erlbaum Associates Ltd., Mahwah, NJ.

---

kmassess *Perform a probabilistic knowledge assessment*

Description

`kmassess` performs a probabilistic knowledge assessment for a given response vector, knowledge structure, and BLIM parameters.

`kmsassess` performs a simplified probabilistic knowledge assessment for a given response vector, knowledge structure, and BLIM parameters. It assumes an equal probability distribution over the knowledge structure as starting point and identical beta and eta values for all items.

Usage

```
kmassess(
  r,
  pks,
  questioning,
  update,
  beta,
  eta,
  zeta0,
  zeta1,
  threshold,
  probdev = FALSE,
  directory = tempdir()
)
```

```
kmsassess(
  r,
  ks,
  questioning,
  update,
  beta,
  eta,
  zeta0,
  zeta1,
  threshold,
  probdev = FALSE,
  directory = NULL
)
```

Arguments

r	Response pattern (binary vector)
pks	Probabilistic knowledge structure: a data frame with a probability distribution in the first columns and the structure matrix in the subsequent columns.
questioning	Questioning rule ("halfsplit" o "informative")
update	Update rule ("Bayesian" or "multiplicative")
beta	Careless error probability
eta	Lucky guess probability
zeta0	Update parameter for wrong responses
zeta1	Update parameter for correct responses
threshold	Probability threshold for stopping criterion
probdev	Provide information on the probability development including Hasse diagrams stored in `tempdir()`. Defaults to FALSE.
directory	Where to store the Hasse diagrams.
ks	Knowledge structure: a binary matrix

Details

`kmassess` implements the stochastic assessment procedures according to Doignon & Falmagne, 1999, chapter 10.

`kmassess` stops if the number of questions has reached twice the number of items.

Value

A list with the following elements:

state Diagnosed knowledge state (binary vector)

probs Resulting probability distribution. If probdev is set to TRUE, a list of probability distributions for each step is given instead.

queried Sequence of items used in the assessment (list)

qtime Average time for finding a question

utime Average time for updating the probabilities

A list with the following elements:

state Diagnosed knowledge state (binary vector)

probs Resulting probability distribution. If probdev is set to TRUE, a list of probability distributions for each step is given instead.

queried Sequence of items used in the assessment (list)

qtime Average time for finding a question

utime Average time for updating the probabilities

Background

Doignon & Falmagne (1985, 1999) proposed knowledge space theory originally with adaptive knowledge assessment in mind. The basic idea is to apply prerequisite relationships between items for reducing the number of problems to be posed to a learner in knowledge assessment.

Falmagne & Doignon (1988; Doignon & Falmange, 1999, chapte 10) proposed a class of stochastic procdures for such adaptive assessment which take into account that careless errors and lucky guesses may happen during the assessment by estimating a probability distribution over the knowledge structure. Such an assessment consists of three important parts

- Question rule
- Update rule
- Stopping criterion

For the **question rule,** they propose the *halfsplit* and the *infomrative* rules, implemented in `kmassesshalfslit` and `kmassessinfomrative`.

For the **update rule,** they again propose two possibilities there the *multiplicative rule* is a generalisation of the (classical) *Bayesian update rule* implemented here in `kmassessmultiplicative` and `kmassessbayesian`, respectively.

As **stopping criterion,** usually a threshold for the maximal probability for one knowledge state is used. It is strongly recommended to keep this larger than 0.5 in order to have one unequivocal resulting state (see also Hockemeyer, 2002).

Framework of assessment functions within the kstMatrix package::

The founding stones are the four aforementioned functions for finding suitable questions and for updating the probability estimates, respectively. They could also be used in an interactive system, e.g. a Shiny app, for "real" adaptive assessment.

The remaining three assessment functions serve for mere simulation of adaptive assessment. `kmassess` takes, among others, a full response pattern as parameter and takes the responses for the selected questions from this vector. `kmsassess` is a simplified version where the update parameters (beta and eta for Bayesian or zeta0 and zeta1 for multiplicative update, respectively) are identical for all items whereas they are item-specific in `kmassess`. Finally, `kmassesssimulation` takes a whole data set, i.e. a collection of response patterns, and does an assessment for each of these patterns. Its result is a data frame which should be suitable for further statistical evaluation,

especially if it is called several times with variant parameters (e.g., structures, update parameters, update and question rules).

Both, `kmsassess` and `kmassesssimulation` call `kmassess`.

Problems:

In rare cases `kmassess` may flip forth and back between probability distributions resulting in an endless loop. Therefore, it stops after twice the number of items delivering a `NULL` result.

References

Doignon, J.-P. & Falmagne, J.-C. (1985). Spaces for the assessment of knowledge. *International Journal of Man-Machne-Studies, 23,* 175-196. doi:10.1016/S00207373(85)800316.

Doignon, J.-P. & Falmagne, J.-C. (1999). *Knowledge Spaces.* Springer Verlag, Berlin. doi:10.1007/9783642586255.

Falmagne, J.-C. & Doignon, J.-P. (1988). A class of stochastic procedures for the assessment of knowledge. *British Journal of Mathematical and Statistical Psychology, 41,* 1-23. doi:10.1111/j.20448317.1988.tb00884.x.

Hoxkemeyer, C. (2002). A comparison of non-deterministic procedures for the adaptive assessment of knowledge. *Psychlogische Beiträge, 44(4),* 495-503.

See Also

Other Knowledge assessment functions: `kmassessbayesian()`, `kmassesshalfsplit()`, `kmassessinformative()`, `kmassessmentsimulation()`, `kmassessmultiplicative()`

Examples

```
kmassess(c(1, 1, 0, 0),
         cbind(as.data.frame(as.matrix(rep(1/9.0, 9), ncol=1)), xpl$space),
         "halfsplit",
         "Bayesian",
         rep(0.12, 4),
         rep(0.1, 4),
         NULL,
         NULL,
         0.55
        )

kmsassess(c(1,1,0,0), xpl$space, "halfsplit", "Bayesian", 0.1, 0.1, NULL, NULL, 0.55)
```

`kmassessbayesian` *Update probability distribution applying Bayesian update*

Description

`kmassessbayesian` updates a probability distribution over a knowledge structure according to the Bayesian update rule.

Usage

```
kmassessbayesian(probs, ks, beta, eta, question, response)
```

Arguments

probs	Probability distribution over the knowledge structure (vector)
ks	Binary matrix of the knowledge structure
beta	Vector of careless error probabilities
eta	Vector of lucky guess probabilities
question	Item that has been posed
response	Correctness of received response (0 or 1)

Value

Updated probability vector

See Also

Other Knowledge assessment functions: kmassess(), kmassesshalfsplit(), kmassessinformative(), kmassessmentsimulation(), kmassessmultiplicative()

Examples

```
kmassessbayesian(c(0.02, 0.1, 0.07, 0.01, 0.4, 0.17, 0.07, 0.08, 0.08),
                 xpl$space,
                 rep(0.2,4),
                 rep(0.1,4),
                 3,
                 1
                )
```

---

kmassesshalfsplit *Determine next question for probabilistic knowledge assessment*

Description

kmassesshalfsplit determines the next question in a probabilistic assessment according to the halfsplit rule.

Usage

```
kmassesshalfsplit(probs, ks)
```

Arguments

probs	Probability distribution over the knowledge structure (vector)
ks	Binary matrix of the knowledge structure

Value

Number of the selected question

See Also

Other Knowledge assessment functions: kmassess(), kmassessbayesian(), kmassessinformative(), kmassessmentsimulation(), kmassessmultiplicative()

Examples

```
kmassesshalfsplit(c(0.02, 0.1, 0.07, 0.01, 0.4, 0.17, 0.07, 0.08, 0.08),
                  xpl$space)
```

---

kmassessinformative *Determine next question for probabilistic knowledge assessment*

Description

kmassessinformative determines the next question in a probabiliststic assessment according to the informative rule.

Usage

```
kmassessinformative(probs, ks, update, beta, eta, zeta0, zeta1)
```

Arguments

probs	Probability distribution over the knowledge structure (vector)
ks	Binary matrix of the knowledge structure
update	Update rule ("Bayesian" or "multiplicative")
beta	Careless error probabilities (vector)
eta	Lucky guess probabilities (vector)
zeta0	Vector of update parameters for wrong responses
zeta1	Vector of update parameters for correct responses

Value

Number of the selected question

See Also

Other Knowledge assessment functions: kmassess(), kmassessbayesian(), kmassesshalfsplit(), kmassessmentsimulation(), kmassessmultiplicative()

Examples

```
kmassessinformative(c(0.02, 0.1, 0.07, 0.01, 0.4, 0.17, 0.07, 0.08, 0.08),
                    xpl$space,
                    "Bayesian",
                    rep(0.3,4),
                    rep(0.2,4),
                    NULL,
                    NULL
                   )
```

---

kmassessmentsimulation

Simulate assessments for a set of response patterns

Description

kmassessmentsimulation does a probabilistic knowledge assessment for each response pattern in a data matrix and stores information about the assessment.

Usage

```
kmassessmentsimulation(
  respdata,
  ks,
  questioning,
  update,
  beta,
  eta,
  zeta0,
  zeta1,
  threshold
)
```

Arguments

respdata	Data matrix
ks	Knowledge structure
questioning	Question rule
update	Updating rule
beta	Careless error probability

eta	Lucky guess probability
zeta0	Update parameter for wrong responses
zeta1	Update parameter for correct responses
threshold	Stopping criterion

Details

kmassessmentsimulation applies the kmsassess function.

Value

Assessment data as data frame

See Also

Other Knowledge assessment functions: kmassess(), kmassessbayesian(), kmassesshalfsplit(), kmassessinformative(), kmassessmultiplicative()

Examples

```
kmassessmentsimulation(
  xpl$data,
  xpl$space,
  "halfsplit",
  "multiplicative",
  NULL,
  NULL,
  5,
  5,
  0.55
)
```

---

kmassessmultiplicative

Update probability distribution applying multiplicative rule

Description

kmassessmultiplicative updates a probability distribution on a knowledge structure according to the multiplicative rule.

Usage

```
kmassessmultiplicative(probs, ks, zeta0, zeta1, question, response)
```

Arguments

probs	Probability distribution over the knowledge structure (vector)
ks	Binary matrix of the knowledge structure
zeta0	Vector of update parameters for wrong responses
zeta1	Vector of update parameters for correct responses
question	Item that has been posed
response	Correctness of received response (0 or 1)

Value

Updated probability vector

See Also

Other Knowledge assessment functions: kmassess(), kmassessbayesian(), kmassesshalfsplit(), kmassessinformative(), kmassessmentsimulation()

Examples

```
kmassessmultiplicative(c(0.02, 0.1, 0.07, 0.01, 0.4, 0.17, 0.07, 0.08, 0.08),
                xpl$space,
                rep(1.2,4),
                rep(2.1,4),
                3,
                1
               )
```

---

kmbasis *Compute the basis of a knowledge space*

Description

kmbasis.kmsurmisefunction takes a surmise function and returns the corresponding basis, i.e. effectively the collection of all clauses in the surmise function.

kmbasis.kmsurmiserelation takes a surmise relation and returns the corresponding basis.

kmbasis.matrix returns a matrix representing the basis of a knowledge space. If x is a knowledge structure or an arbitrary family of sets kmbasis returns the basis of the smallest knowledge space containing x.

Usage

```
kmbasis(x)

## S3 method for class 'kmsurmisefunction'
kmbasis(x)

## S3 method for class 'kmsurmiserelation'
kmbasis(x)

## S3 method for class 'matrix'
kmbasis(x)
```

Arguments

x	Binary matrix representing a knowledge space

Value

Binary matrix representing the basis of the corresponding knowledge space.

See Also

Other Different representations for knowledge structures: `kmspace()`, `kmsurmisefunction()`, `kmsurmiserelation()`

Examples

```
kmbasis(xpl$sf)


kmbasis(xpl$sr)

kmbasis(xpl$space)
```

---

kmbasisdiagram *DEPRECATED:* `kmbasisdiagram()` *has been replaced by a* `plot()` *method.*

Description

DEPRECATED: `kmbasisdiagram()` has been replaced by a `plot()` method.

Usage

```
kmbasisdiagram(x, horizontal = TRUE, colors = NULL)
```

Arguments

x	Basis
horizontal	Boolean whether the diagram should be drawn horizontally or vertically (default TRUE)
colors	Color vector (default NULL)

See Also

Other Deprecated functions and methods: kmSF2basis(), kmSR2basis(), kmSRdiagram(), kmhasse()

---

kmbasisfringe *Compute the fringe of a state within a knowledge structure using its basis*

Description

kmbasisfringe computes the fringe of a state within a knowledge structure, i.e. the set of items by which the state differs from its neighbours.

Usage

```
kmbasisfringe(state, basis)

kmbasisinnerfringe(state, basis)

kmbasisouterfringe(state, basis)
```

Arguments

state	Binary vector representing a knowledge state
basis	kmbasis object

Value

Binary vector representing the fringe

References

Hockemeyer C (1997). Using the Basis of a Knowledge Space for Determining the Fringe of a Knowledge State. *Journal of Mathematical Psychology, 41,* 275–279.

See Also

Other Functions for fringes & neighbourhoods: kmbasisneighbourhood(), kmfringe(), kmneighbourhood()

Examples

```
kmbasisfringe(c(1,0,0,0), xpl$basis)
```

kmbasisneighbourhood *Compute the neighbourhod of a state within a knowledge structure using its basis*

Description

kmbasisneighbourhood computes the neighbourhood of a state within a knowledge structure, i.e. the family of all other states with a symmetric set difference of 1.

Usage

```
kmbasisneighbourhood(state, basis, include = FALSE)
```

Arguments

state	Binary vector representing a knowledge state
basis	kmbasis object
include	Boolean whether the original state should be included in the result (default FALSE)

Value

Matrix containing the neighbouring states, one per row

See Also

Other Functions for fringes & neighbourhoods: kmbasisfringe(), kmfringe(), kmneighbourhood()

Examples

```
kmbasisneighbourhood(c(1,1,0,0), xpl$basis)
```

kmclosure *Determine the closure of KST objects*

Description

kmclosure() is an S3 method closing kstMatrix objects.

Usage

```
kmclosure(x, ...)

## S3 method for class 'kmattributionfunction'
kmclosure(x, ...)

## S3 method for class 'kmattributionrelation'
kmclosure(x, ...)

## S3 method for class 'kmdata'
kmclosure(x, ...)

## S3 method for class 'kmfamset'
kmclosure(x, ..., closure = "union")
```

Arguments

x	KST object
...	Optional additional parameters
closure	Type of closure For families of sets (kmfamset), kmclosure is basically a wrapper for kmunionclosure() and kmintersectionclosure. closure() may take the values "union" (default) or "intersection". The former requests a closure under union, the latter a closure under union **and** intersection.

Value

Corresponding closed KST object

An attribution relation or attribution function is closed to the corresponding surmise relation or surmise function, respectively.

A family of sets (including bases, knowledge structure, and knowledge space) is closed to a knowledge space or quasi-ordinal knowledge space.

See Also

Other Closure operators: kmintersectionclosure(), kmunionclosure()

---

kmcolors *Determine a color vector based on probabilities*

Description

kmcolors takes a probabilty vector and a color palette and creates a color vector to be used with kstMatrix::plot.

Usage

```
kmcolors(prob, palette = cm.colors)
```

Arguments

prob	Probability vector
palette	Color palette (default = cm.colors)

See Also

Other Functions for plotting and printing knowledge structures: kmprettyprint(), plot

---

kmdist *Compute the distance between a data set and a knowledge structure*

Description

kmdist returns a named vector with the frequencies of distances between a set of response patterns and a knowledge structure. This vector can be used to compute, e.g., the Discrepancy Index (DI) or the Distance Agreement Coefficient (DA).

Usage

```
kmdist(data, struct)
```

Arguments

data	Binary matrix representing a set of response patterns
struct	Binary matrix representing a knowledge structure

Value

Distance distribution vector

See Also

Other Functions for working with data: kmSRvalidate(), kmgenerate(), kmiita2SR(), kmsimulate(), kmvalidate()

Examples

```
kmdist(xpl$data, xpl$space)
```

kmdoubleequal *Test two double numbers on equity with a certain tolerance*

Description

Test two double numbers on equity with a certain tolerance

Usage

```
kmdoubleequal(x, y, tol = sqrt(.Machine$double.eps))
```

Arguments

x	First double to compare
y	Second double to compare
tol	Tolerance optional)

Value

Boolean for (approximate) equity

See Also

Other Utilities: binary_matrix_product(), kmminimalfamset(), kmsetiselement(), kmsymmsetdiff(), kmtrivial

Examples

```
kmdoubleequal(0.5+0.5, 1)
0.5 + 0.5 == 1
```

kmeqreduction *Reduce a family of knowledge states with respect to item equivalence*

Description

kmeqreduction takes a family of knowledge states and returns its reduction to non-equivalent items.

Usage

```
kmeqreduction(x)
```

Arguments

x	Family of sets (kmfamset object)

Value

Binary matrix reduced by equivalences

See Also

Other Functions for working with knowledge structures: kmexpand(), kmintersection(), kmmesh(), kmnotions(), kmsubstructure(), kmtrivial, kmunion()

Examples

```
kmeqreduction(xpl$space)
```

---

kmexpand *Expand an attribution function to a larger set of items and close it to a surmise function*

Description

kmexpand() expands an attribution function by additional items and closes it to a surmise function.

Usage

```
kmexpand(af, ni)
```

Arguments

af	Attribution function
ni	New (complete) item list

Value

Expanded surmise function

See Also

Other Functions for working with knowledge structures: kmeqreduction(), kmintersection(), kmmesh(), kmnotions(), kmsubstructure(), kmtrivial, kmunion()

Examples

```
xpl$sf
kmexpand(xpl$sf, c("a", "b", "c", "d", "e", "f"))
```

kmfamset *Convert a binary matrix to a* kmfamset *object (family of sets)*

Description

kmfamset returns a kmfamset object after checking that the passed object is a binary matrix with all different rows. If the passe object inherits the kmfamset property, nothing else is changed.

Usage

```
kmfamset(x)
```

Arguments

x	Binary matrix representing a family of sets

Value

kmfamset object

See Also

Other Constructors: kmqspace(), kmspace(), kmstructure()

Examples

```
m <- as.matrix(c(1,0,0,0,1,0,1,1,1), nrow=3, byrow=TRUE)
kmfamset(m)
```

kmfringe *Compute the fringe of a state within a knowledge structure*

Description

kmfringe computes the fringe of a state within a knowledge structure, i.e. the set of items by which the state differs from its neighbours.

Usage

```
kmfringe(state, struct)

kminnerfringe(state, struct)

kmouterfringe(state, struct)
```

Arguments

state	Binary vector representing a knowledge state
struct	Binary matrix representing a knowledge structure

Value

Binary vector representing the fringe

See Also

Other Functions for fringes & neighbourhoods: kmbasisfringe(), kmbasisneighbourhood(), kmneighbourhood()

Examples

```
kmfringe(c(1,0,0,0), xpl$space)
```

---

kmgenerate *Generate a knowledge structure from a set of response patterns*

Description

kmgenerate returns a matrix representing a knowledge structure generated from data. It uses a simplistic approach: patterns with a frequency above a specified threshold are considered as knowledge states. If the specified threshold is 0 (default) a real threshold is computed as N (number of response patterns) divided by 2^|Q|. Please note that the number of response patterns should be much higher than the size of the power set of the item set Q. A factor of art least 10 is recommended. Currently, the number of items is limited to the number of bits in a C long minus one (i.e. 31 under Windows and 63 otherwise). But we would probably run into memory problems way earlier anyway.

Usage

```
kmgenerate(x, threshold = 0)
```

Arguments

x	Binary matrix representing a data set
threshold	Threshold for taking response patterns as knowledge states (default 0)

Value

Binary matrix representing the generated knowledge structure

See Also

Other Functions for working with data: kmSRvalidate(), kmdist(), kmiita2SR(), kmsimulate(), kmvalidate()

Examples

```
kmgenerate(xpl$sim, 15)
```

`kmgradations` *Determine all gradations between two states*

Description

Determine all gradations between two states

Usage

```
kmgradations(structure, from = NULL, to = NULL)
```

Arguments

`structure`	Knowledge structure
`from`	Starting state (if NULL (default), it is the empty set)
`to`	Goal state (if NULL (default), it is the full item set)

Value

A list of gradations where each gradation is a list of states

See Also

Other Functions for wellgradedness & learning paths: `kmiswellgraded()`, `kmlearningpathmatrices()`, `kmlearningpaths()`

Examples

```
kmgradations(xpl$space)
```

kmhasse *DEPRECATED:* kmhasse() *has been replaced by a* plot() *method.*

Description

DEPRECATED: kmhasse() has been replaced by a plot() method.

Usage

```
kmhasse(x, horizontal = TRUE, colors = NULL)
```

Arguments

x	Knowledge structure
horizontal	Boolean whether the diagram should be drawn horizontally or vertically (default TRUE)
colors	Color vector (default NULL)

See Also

Other Deprecated functions and methods: kmSF2basis(), kmSR2basis(), kmSRdiagram(), kmbasisdiagram()

kmheights *Compute the heights of items in a knowledge space specified through its basis*

Description

The height of an item in the knowledge space is defined as the minimal cardinality of the states containing it minus 1, i.e. the minimal number of prerequisites.

Usage

```
kmheights(b)
```

Arguments

b	Basis

Value

Data.frame with a table of heights.

See Also

Other Properties of knowledge structures: kmiswellgraded(), kmnotions()

Examples

```
kmheights(xpl$basis)
```

`kmiita2SR` *Convert an IITA result into a surmise relation matrix*

Description

`kmiita2SR` takes the result of a `DAKS::iita()` call and delivers the matrix of the computed surmise relation.

Usage

```
kmiita2SR(ii, names = NULL, items = 0)
```

Arguments

`ii`	`iita()` result
`names`	Vector of item names (default NULL)
`items`	Minimal number of items (default 0)

Value

Surmise relation matrix

The `iita()` function looses information on the item names and uses consecutive numbers instead. The `implications` part of its result does not give any hint on isolated items, i.e. items which neither have a prerequisite nor are prerequisite of any other item. Therefore, a minimal number of items can be passed to `kmiita2SR()`. If the highest item number within `implications` is higher, this `items` parameter is ignored.

See Also

Other Functions for working with data: `kmSRvalidate()`, `kmdist()`, `kmgenerate()`, `kmsimulate()`, `kmvalidate()`

`kmintersection` *Compute the "intersection" of two KST objects*

Description

Both objects must be of the same kstMatrix class, the result will be of the same class, i.e. a respective closure operator will be executed automatically.

Please note that the intersection of surmise relations is automatically a surmise relation without any closure needs.

Usage

```
kmintersection(x, y)

## S3 method for class 'kmqspace'
kmintersection(x, y)

## S3 method for class 'kmspace'
kmintersection(x, y)

## S3 method for class 'kmsurmisefunction'
kmintersection(x, y)

## S3 method for class 'kmsurmiserelation'
kmintersection(x, y)
```

Arguments

`x`	First KST object
`y`	Second KST object

Value

Intersection of `x` and `y`

See Also

Other Functions for working with knowledge structures: `kmeqreduction()`, `kmexpand()`, `kmmesh()`, `kmnotions()`, `kmsubstructure()`, `kmtrivial`, `kmunion()`

Examples

```
sf <- kmsurmisefunction(kmbasis(xpl$sr))
kmintersection(xpl$sf, sf)
```

kmintersectionclosure *Close a knowledge space under intersection*

Description

Close a knowledge space under intersection

Usage

```
kmintersectionclosure(x)
```

Arguments

x	Knowledge space

Value

Quasi–ordinal knowledge space

See Also

Other Closure operators: `kmclosure()`, `kmunionclosure()`

Examples

```
kmintersectionclosure(xpl$space)
```

kmiswellgraded *Check for wellgradedness of a knowledge structure*

Description

`kmiswellgraded()` returns whether a knowledge space (or its basis) is well-graded.

Usage

```
kmiswellgraded(x)
```

Arguments

x	`kmspace` or `kmbasis`

Value

Logical value specifying whether `x` is well-graded

References

Doignon, J.-P. & Falmagne, J.-C. (1999). *Knowledge Spaces.* Springer–Verlag, Berlin.

See Also

Other Properties of knowledge structures: `kmheights()`, `kmnotions()`

Other Functions for wellgradedness & learning paths: `kmgradations()`, `kmlearningpathmatrices()`, `kmlearningpaths()`

Examples

```
kmiswellgraded(xpl$space)
```

`kmlearningpathmatrices` *Determine all learning paths in a knowledge structure*

Description

Determine all learning paths in a knowledge structure

Usage

```
kmlearningpathmatrices(structure, through = NULL, from = NULL, to = NULL)
```

Arguments

`structure`	Knowledge structure
`through`	Knowledge state through which the paths should go
`from`	Knowledge state where paths should start
`to`	Knowledge state where paths should end

Value

A list of learning paths where each learning path matrix

If `through` is specified, only those learning paths are listed which go through that state (binary vector). If `from` and `to` are specified, the respective sub paths are returned.

See Also

Other Functions for wellgradedness & learning paths: `kmgradations()`, `kmiswellgraded()`, `kmlearningpaths()`

Examples

```
kmlearningpathmatrices(xpl$space)
```

`kmlearningpaths` *Determine all learning paths in a knowledge structure*

Description

Determine all learning paths in a knowledge structure

Usage

```
kmlearningpaths(
  structure,
  through = NULL,
  from = NULL,
  to = NULL,
  simplified = FALSE
)
```

Arguments

`structure`	Knowledge structure
`through`	Knowledge state through which the paths should go
`from`	Knowledge state where paths should start
`to`	Knowledge state where paths should end
`simplified`	Simplified printing of states within the paths (default FALSE)

Value

A list of learning paths where each learning path is a list of states

If `through` is specified, only those learning paths are listed which go through that state (binary vector). If `from` and `to` are specified, the respective sub paths are returned.

See Also

Other Functions for wellgradedness & learning paths: `kmgradations()`, `kmiswellgraded()`, `kmlearningpathmatrices()`

Examples

```
kmlearningpaths(xpl$space)
```

kmmesh *Determine the maximal mesh of two knowledge structures*

Description

If the two structures are not compatible, NULL is returned.

Usage

```
kmmesh(x, y)
```

Arguments

x	First structure
y	Second structure

Value

Maximal mesh

References

Falmagne J, Doignon J (1998). "Meshing Knowledge Structures." In Dowling CE, Roberts FS, Theuns P (eds.), *Recent Progress in Mathematical Psychology,* 143–153. Lawrence Erlbaum Associates Ltd., Mahwah, NJ.

See Also

Other Functions for working with knowledge structures: kmeqreduction(), kmexpand(), kmintersection(), kmnotions(), kmsubstructure(), kmtrivial, kmunion()

kmminimalfamset *Determine subfamily of minimal states in a* kmfamset

Description

Determine subfamily of minimal states in a kmfamset

Usage

```
kmminimalfamset(x)
```

Arguments

x	kmfamset

Value

kmfamset of minimal states

`kmminimalfamset()` determines the sub family of minimal sets within a famset. If x is a knowledge structure, the result will be a family containing (only) the empty set because the empty set is a state in any knowledge structure and is its minimal state.

The concept of the minimal state of a family of sets should not be confused with the concept of knowledge states minimal for some item.

See Also

Other Utilities: `binary_matrix_product()`, `kmdoubleequal()`, `kmsetiselement()`, `kmsymmsetdiff()`, `kmtrivial`

Examples

```
m <- matrix(c(1,0,0,0,1,0,1,0,1), ncol=3, byrow=TRUE)
m
kmminimalfamset(kmfamset(m))
```

---

kmneighbourhood *Compute the neighbourhod of a state within a knowledge structure*

Description

`kmneighbourhood` computes the neighbourhood of a state within a knowledge structure, i.e. the family of all other states with a symmetric set difference of 1.

`kmnneighbourhood` computes the n-neighbourhood of a state within a knowledge structure, i.e. the family of all other states with a symmetric set difference maximal n.

Usage

```
kmneighbourhood(state, struct, include = FALSE)

kmnneighbourhood(state, struct, distance, include = FALSE)
```

Arguments

`state`	Binary vector representing a knowledge state
`struct`	Binary matrix representing a knowledge structure
`include`	Boolean whether the original state should be included (defaul FALSE)
`distance`	Size of the n-neighbourhood

Value

Matrix containing the neighbouring states, one per row

Matrix containing the neighbouring states, one per row

See Also

Other Functions for fringes & neighbourhoods: `kmbasisfringe()`, `kmbasisneighbourhood()`, `kmfringe()`

Examples

```
kmneighbourhood(c(1,1,0,0), xpl$space)

kmnneighbourhood(c(1,1,0,0), xpl$space, 2)
```

---

kmnotions *Determine the notions of a knowledge structure*

Description

`kmnotions` returns a matrix representing the notions of a knowledge structure.

Usage

```
kmnotions(x)
```

Arguments

`x`	Binary matrix representing a knowledge structure

Value

Binary matrix representing notions in the knowledge structure

The matrix has a '1' in row 'i' and column 'j' if 'i' and 'j' belong to the same notion (i.e. are equivalent). It is a symmetric matrix with '1's in the main diagonal.

See Also

Other Properties of knowledge structures: `kmheights()`, `kmiswellgraded()`

Other Functions for working with knowledge structures: `kmeqreduction()`, `kmexpand()`, `kmintersection()`, `kmmesh()`, `kmsubstructure()`, `kmtrivial`, `kmunion()`

Examples

```
kmnotions(xpl$space)
```

kmprettyprint *Print KST objects in set notation*

Description

kmprettyprint() formats KST objects in set notation for printing. For further details, see below.

Usage

```
kmprettyprint(
  x,
  simplified = FALSE,
  ...,
  open = "{",
  close = "}",
  inneropen = "{",
  innerclose = "}"
)

## S3 method for class 'numeric'
kmprettyprint(
  x,
  simplified = FALSE,
  ...,
  open = "{",
  close = "}",
  itemsep = ", "
)

## S3 method for class 'kmfamset'
kmprettyprint(
  x,
  simplified = FALSE,
  ...,
  open = "{",
  close = "}",
  inneropen = "{",
  innerclose = "}"
)

## S3 method for class 'kmattributionfunction'
kmprettyprint(
  x,
  simplified = FALSE,
  ...,
  open = "[",
  close = "]",
```

```
  inneropen = "{",
  innerclose = "}",
  mapsto = NULL
)

## S3 method for class 'kmattributionrelation'
kmprettyprint(
  x,
  simplified = FALSE,
  ...,
  open = "{",
  close = "}",
  inneropen = "(",
  innerclose = ")",
  tuple = TRUE,
  relation = NULL
)

## S3 method for class 'kmdata'
kmprettyprint(
  x,
  simplified = FALSE,
  ...,
  open = "(",
  close = ")",
  inneropen = "{",
  innerclose = "}"
)

## S3 method for class 'kmlearningpathlist'
kmprettyprint(
  x,
  simplified = FALSE,
  ...,
  open = "",
  close = "",
  inneropen = "{",
  innerclose = "}",
  path = NULL
)

## S3 method for class 'kmlearningpathmatrices'
kmprettyprint(
  x,
  simplified = FALSE,
  ...,
  open = "",
  close = "",
```

```
  inneropen = "{",
  innerclose = "}",
  path = NULL
)

## S3 method for class 'kmlearningpathmatrix'
kmprettyprint(
  x,
  simplified = FALSE,
  ...,
  open = "",
  close = "",
  inneropen = "{",
  innerclose = "}",
  path = NULL
)
```

Arguments

x	Object to be printed
simplified	Omit inner braces for families of sets (default FALSE)
...	Optional further arguments
open	(Outer) opening bracket
close	(Outer) closing bracket
inneropen	Inner opening bracket
innerclose	Inner closing bracket
itemsep	Item separating character string
mapsto	Mapsto symbol for pretty-printing surmise and attribution functions (Default: '$\mapsto$')
tuple	Should we use tuple notation (default TRUE)?
relation	Relation symbol for pretty-printing surmise and attribution relations
path	Rightarrow for pretty-printing path (Default: '$\rightarrow$')

Details

If simplified is TRUE, a state '{a, b, c}' is printed as 'abc'. This should only be used if the item names are single letters.

For Families of sets (kmfamset and sub classes), the states are ordered by cardinality.

inneropen and innerclose are ignored for numeric vectors (i.e. states/plain sets).

mapsto defines the 'mapsto' symbol for pretty-printing a surmise or attribution function. It may contain more than one character, as in the default.

For surmise and attribution relations, there are two special parameters. tuple sets whether the relation shall be printed as set of ordered pairs or whether it shall be printed as tuple of pairs with relation symbol which can be set with the relation parameter.

For kmlearningpaths lists, thre result should be printed through cat() instead of print() because the latter does not execute the line breaks.

Please note that the methods can also be used for sub classes, e.g. the kmprettyprinting.kmfamset() method is also applicable for bases, knowledge structures, and knowledge spaces.

kmprettyprint() was decidedly not implemented as print() method. This way, one can *print* KST objects in matrix or data frame notation or *prettyprint* them in set notation etc.

See Also

Other Functions for plotting and printing knowledge structures: kmcolors(), plot

Examples

```
kmprettyprint(xpl$basis)
kmprettyprint(xpl$basis, simplified=TRUE)
kmprettyprint(xpl$sr)
kmprettyprint(xpl$sf)
```

---

kmqspace *Obtain a quasi-ordinal knowledge space*

Description

kmqspace() computes the quasi-ordinal space corresponding to a given KST object.

kmqspace() computes the smallest quasi-ordinal space containing a given family of sets.

kmqspace.kmsurmisefunction() takes a surmise function and determines the quasi-ordinal knowledge space which is the closure under intersection of the knowledge space delineated by this surmise function.

kmqspace.kmsurmiserelation() computes the quasi-ordinal knowledge space delineated by a surmise relation

Usage

```
kmqspace(x)

## S3 method for class 'kmfamset'
kmqspace(x)

## S3 method for class 'kmsurmisefunction'
kmqspace(x)

## S3 method for class 'kmsurmiserelation'
kmqspace(x)
```

Arguments

x	KST object

Value

Corresponding kmqspace object

See Also

Other Constructors: kmfamset(), kmspace(), kmstructure()

Examples

```
m <- kmfamset(matrix(c(1,0,0,0,1,0,1,1,1), nrow=3, byrow=TRUE))
kmqspace(m)

m <- kmfamset(matrix(c(1,0,0,0,1,0,1,1,1), nrow=3, byrow=TRUE))
kmqspace(m)

sr <- kmsurmiserelation(xpl$space)
kmqspace(sr)
```

---

kmrefine *Refine a famset by dissolving a notion through a basis on the notion*

Description

Refine a famset by dissolving a notion through a basis on the notion

Usage

```
kmrefine(x, b)
```

Arguments

x	Family of sets
b	Basis for refinement

Value

Refined family of sets

Examples

```
x <- kmfamset(matrix(c(1,0,0,0,0,1,0,0,1,1,1,1), ncol=4, byrow=TRUE))
colnames(x) <- c("a", "b", "c", "d")
x
b <- kmbasis(kmfamset(matrix(c(1,0,1,1), ncol=2, byrow=TRUE)))
colnames(b) <- c("c", "d")
b
kmrefine(x,b)
```

`kmsetiselement` *Test if a state is contained in a family of states*

Description

Test if a state is contained in a family of states

Usage

```
kmsetiselement(s, f)
```

Arguments

`s`	State
`f`	Family of sets

Value

Boolean is s is contained in f

See Also

Other Utilities: `binary_matrix_product()`, `kmdoubleequal()`, `kmminimalfamset()`, `kmsymmsetdiff()`, `kmtrivial`

Examples

```
kmsetiselement(c(1,1,1,0), xpl$space)
```

kmSF2basis *DEPRECATED:* kmSF2basis() *has been replaced by* kmbasis().

Description

DEPRECATED: kmSF2basis() has been replaced by kmbasis().

Usage

```
kmSF2basis(x)
```

Arguments

x	Surmise function

Value

Corresponding basis

See Also

Other Deprecated functions and methods: kmSR2basis(), kmSRdiagram(), kmbasisdiagram(), kmhasse()

kmsimulate *Simulate a set of response patterns according to the BLIM*

Description

kmsimulate returns a data set of n simulated response patterns based on the knowledge structure x given as a binary matrix. The simulation follows the BLIM (Basic Local Independence Model; see Doigon & Falmagne, 1999).

Usage

```
kmsimulate(x, n, beta, eta)
```

Arguments

x	Binary matrix representing a knowledge space
n	Number of simulated response patterns
beta	Careless error probability value or vector
eta	Lucky guess probability value or vector

Details

The `beta` and `eta` parameters must be either single numericals or vectors with a length identical to the number of rows in the `x` matrix. A mixture is possible.

The `sample` function used by `kmsimulate` might work inaccurately for knowledge structures `x` with 2^31 or more states.

Value

Binary matrix representing the simulated data set

References

Doignon, J.-P. & Falmagne, J.-C. (1999). *Knowledge Spaces.* Springer–Verlag, Berlin.

See Also

Other Functions for working with data: `kmSRvalidate()`, `kmdist()`, `kmgenerate()`, `kmiita2SR()`, `kmvalidate()`

Examples

```
kmsimulate(xpl$space, 50, 0.2, 0.1)
kmsimulate(xpl$space, 50, c(0.2, 0.25, 0.15, 0.2), c(0.1, 0.15, 0.05, 0.1))
kmsimulate(xpl$space, 50, c(0.2, 0.25, 0.15, 0.2), 0)
```

---

kmspace `kmspace` *constructor method*

Description

`kmspace()` determines the knowledge space corresponding to a KST object

Please note that a knowledge space delineated by a surmise relation is quasi-ordinal. Therefore, the result is automatically marked as `kmqspace` object.

Usage

```
kmspace(x)

## S3 method for class 'kmdata'
kmspace(x)

## S3 method for class 'kmfamset'
kmspace(x)

## S3 method for class 'kmsurmisefunction'
kmspace(x)
```

```
## S3 method for class 'kmsurmiserelation'
kmspace(x)
```

Arguments

x	KST object

Value

Corresponding knowledge space

See Also

Other Constructors: `kmfamset()`, `kmqspace()`, `kmstructure()`

Other Different representations for knowledge structures: `kmbasis()`, `kmsurmisefunction()`, `kmsurmiserelation()`

Examples

```
kmspace(xpl$basis)
kmspace(xpl$sf)
```

---

kmSR2basis *DEPRECATED:* `kmSR2basis()` *has been replaced by* `kmbasis()`.

Description

DEPRECATED: `kmSR2basis()` has been replaced by `kmbasis()`.

Usage

```
kmSR2basis(x)
```

Arguments

x	Surmise relation

Value

Corresponding basis

See Also

Other Deprecated functions and methods: `kmSF2basis()`, `kmSRdiagram()`, `kmbasisdiagram()`, `kmhasse()`

kmSRdiagram — *DEPRECATED:* kmSRdiagram() *has been replaced by a* plot() *method.*

Description

DEPRECATED: kmSRdiagram() has been replaced by a plot() method.

Usage

```
kmSRdiagram(x, horizontal = FALSE, colors = NULL)
```

Arguments

x	Surmise relation
horizontal	Boolean whether the diagram should be drawn horizontally or vertically (default FALSE)
colors	Color vector (default NULL)

See Also

Other Deprecated functions and methods: kmSF2basis(), kmSR2basis(), kmbasisdiagram(), kmhasse()

kmSRvalidate — *Validate a surmise relation against a data set*

Description

kmSRvalidate returns a list with two elements, Goodman & Kruskal's gamma value and the violational coefficient (VC).

Usage

```
kmSRvalidate(data, sr)
```

Arguments

data	Binary matrix representing a set of response patterns
sr	Binary matrix representing a surmise relation

Value

A list with two elements:

gamma Goodman & Kruskal's gamma index

vc Violational Coefficient

See Also

Other Functions for working with data: kmdist(), kmgenerate(), kmiita2SR(), kmsimulate(), kmvalidate()

Examples

```
kmSRvalidate(xpl$data, xpl$sr)
```

---

kmstructure *Convert a binary matrix to a* kmstructure *object*

Description

kmstructure() returns a kmstructure object after checking that the passed object is a binary matrix without double rows. The empty set and the full item set are added if missing.

Usage

```
kmstructure(x)
```

Arguments

x	Binary matrix representing a family of sets

Value

kmstructure object

See Also

Other Constructors: kmfamset(), kmqspace(), kmspace()

Examples

```
m <- matrix(c(1,0,0,0,1,0,1,1,1), nrow=3, byrow=TRUE)
kmstructure(m)
```

`kmsubstructure` *Reduce a KST object to a subset of items*

Description

Reduce a KST object to a subset of items

Usage

```
kmsubstructure(x, items)

## S3 method for class 'kmdata'
kmsubstructure(x, items)

## S3 method for class 'kmbasis'
kmsubstructure(x, items)

## S3 method for class 'kmstructure'
kmsubstructure(x, items)

## S3 method for class 'kmspace'
kmsubstructure(x, items)

## S3 method for class 'kmsurmiserelation'
kmsubstructure(x, items)

## S3 method for class 'kmsurmisefunction'
kmsubstructure(x, items)
```

Arguments

`x`	KST object
`items`	Selection of items (vector) to which the KST object shall be reduced

Value

Sub structure KST object (the kstMatrix class and its properties are preserved)

`kmsubstructure()` takes a KST object and determines a sub-structure based on the list of items.

The items can be specified through their item names or through the respective column numbers. At least two items must be specified.

See Also

Other Functions for working with knowledge structures: `kmeqreduction()`, `kmexpand()`, `kmintersection()`, `kmmesh()`, `kmnotions()`, `kmtrivial`, `kmunion()`

Examples

```
phsg$basis
kmsubstructure(phsg$basis, c("I1", "I2", "I5"))
```

`kmsurmisefunction` *Compute the surmise function for a knowledge space or basis*

Description

`kmsurmisefunction` returns a data frame representing the surmise function for a knowledge space or basis. The rows of the data frame are ordered by item name.

Usage

```
kmsurmisefunction(x, itemname = "Item")
```

Arguments

`x`	Binary matrix representing a knowledge space or basis
`itemname`	Name for the first column of the resulting data frame

Value

Data frame representing the surmise unction of `x`.

See Also

Other Different representations for knowledge structures: `kmbasis()`, `kmspace()`, `kmsurmiserelation()`

Examples

```
kmsurmisefunction(xpl$space)
```

kmsurmiserelation *Compute the surmise relation of a quasi-ordinal knowledge space*

Description

kmsurmiserelation returns a matrix representing the surmise relation of a quasi-ordinal knowledge space. If x is a general knowledge space, a knowledge structure or an arbitrary family of sets, kmsurmiserelation returns the surmise relation of the smallest quasi-ordinal knowledge space containing x.

Usage

```
kmsurmiserelation(x)
```

Arguments

x	Binary matrix representing a family of sets (class kmfamset)

Value

Binary matrix representing the surmise relation

Note: The columns of the surmise relation matrix describe the minimal state for the respective item in the quasi-ordinal knowledge space.

See Also

Other Different representations for knowledge structures: kmbasis(), kmspace(), kmsurmisefunction()

Examples

```
kmsurmiserelation(xpl$space)
```

kmsymmsetdiff *Compute the symmetric set difference between two sets*

Description

Compute the symmetric set difference between two sets

Usage

```
kmsymmsetdiff(x, y)

kmsetdistance(x, y)
```

Arguments

x	Binary vector representing a set
y	Binary vector representing a set

Value

kmsymmsetdiff: Symmetric set difference between 'x' and 'y'

kmsetdistance: Distance between the sets 'x' and 'y', i.e. the cardinality of the symmetric set difference

See Also

Other Utilities: `binary_matrix_product()`, `kmdoubleequal()`, `kmminimalfamset()`, `kmsetiselement()`, `kmtrivial`

Examples

```
kmsymmsetdiff(c(1,0,0), c(1,1,0))

kmsetdistance(c(1,0,0), c(1,1,0))
```

---

kmtrivial *Create trivial knowledge spaces*

Description

These functions create trivial knowledge spaces of a given item number. The minimal space contains just the empty set and the full item set while the maximal space is equal to the power set.

Usage

```
kmminimalspace(noi)

kmmaximalspace(noi)
```

Arguments

noi	Number of items

Details

Please note that the computation time for creating large power sets can grow quite large easily.

Value

A binary matrix representing the respective knowledge space

See Also

Other Utilities: binary_matrix_product(), kmdoubleequal(), kmminimalfamset(), kmsetiselement(), kmsymmsetdiff()

Other Functions for working with knowledge structures: kmeqreduction(), kmexpand(), kmintersection(), kmmesh(), kmnotions(), kmsubstructure(), kmunion()

Examples

```
kmminimalspace(3)
kmmaximalspace(3)
```

---

kmunion *Compute the "union" of two KST objects*

Description

Both objects must be of the same kstMatrix class, the result will be of the same class, i.e. a respective closure operator will be executed automatically.

Usage

```
kmunion(x, y)

## S3 method for class 'kmfamset'
kmunion(x, y)

## S3 method for class 'kmbasis'
kmunion(x, y)

## S3 method for class 'kmspace'
kmunion(x, y)

## S3 method for class 'kmstructure'
kmunion(x, y)

## S3 method for class 'kmsurmisefunction'
kmunion(x, y)

## S3 method for class 'kmsurmiserelation'
kmunion(x, y)
```

Arguments

x	First object
y	Second object

Details

For surmise functions, the algorithm by Dowling & Hockemeyer (1998) is used.

Please note that, for families of sets, there are the functions `kmunionclosure()` and `kmintersectionclosure()`.

Value

Union of `x` and `y`

References

Dowling CE, Hockemeyer C (1998). "Computing the Intersection of Knowledge Spaces Using only their Basis." In Dowling CE, Roberts FS, Theuns P (eds.), *Recent Progress in Mathematical Psychology,* 133–141. Lawrence Erlbaum Associates Ltd., Mahwah, NJ.

See Also

Other Functions for working with knowledge structures: `kmeqreduction()`, `kmexpand()`, `kmintersection()`, `kmmesh()`, `kmnotions()`, `kmsubstructure()`, `kmtrivial`

---

kmunionclosure *Close a family of sets under union, i.e. determine the smallest knowledge space containing it*

Description

Close a family of sets under union, i.e. determine the smallest knowledge space containing it

Usage

```
kmunionclosure(x)
```

Arguments

`x`	Family of set

Value

Knowledge space

`kmunionclosure` implements the irredundant algorithm developed by Dowling (1993).

References

Dowling, C. E. (1993). On the irredundant construction of knowledge spaces. *Journal of Mathematical Psychology, 37,* 49–62.

See Also

Other Closure operators: `kmclosure()`, `kmintersectionclosure()`

Examples

```
kmunionclosure(xpl$basis)
```

`kmvalidate` *Validate a knowledge structure against a data set*

Description

`kmvalidate` returns a list with three elements, a named vector (dist) with the frequencies of distances between a set of response patterns and a knowledge structure, the Discrepancy Index (DI), and the Distance Agreement Coefficient (DA).

Usage

```
kmvalidate(data, struct)
```

Arguments

`data`	Binary matrix representing a set of response patterns
`struct`	Binary matrix representing a knowledge structure

Value

A list with three elements:

dist Distance distribution vector

DI Discrepancy Index

DA Distance Agreement Coefficient

Warning

The DA computation can take quite some time for larger item sets as the power set has to be computed. For item sets with around 30 items or more, it may even crash the system due to huge memory requests.

See Also

Other Functions for working with data: `kmSRvalidate()`, `kmdist()`, `kmgenerate()`, `kmiita2SR()`, `kmsimulate()`

Examples

```
kmvalidate(xpl$data, xpl$space)
```

phsg *Knowledge space on linear functions*

Description

Basis of a small knowledge space and list of items on linear functions used in a manuscript by Steiner et al.

Usage

```
phsg
```

Format

A list containing the basis and the list of items

plot *Plot a Hasse diagram*

Description

`plot` draws a Hasse diagram for a family of sets or a surmise relation.

Usage

```
## S3 method for class 'kmfamset'
plot(
  x,
  ...,
  state = NULL,
  horizontal = FALSE,
  colors = NULL,
  keepNames = TRUE,
  itemsep = ",",
  braces = TRUE,
  vertexshape = "oval",
  arrowhead = "none",
  arrowtail = "none",
  edgelabel = FALSE,
  verbose = 0
)

## S3 method for class 'kmneighbourhood'
plot(
  x,
```

```
  ...,
  horizontal = FALSE,
  colors = c("#eeee00", "#aaccff", "#bbffbb"),
  keepNames = TRUE,
  itemsep = ",",
  braces = TRUE,
  vertexshape = "oval",
  arrowhead = "none",
  arrowtail = "none",
  edgelabel = FALSE,
  state = NULL,
  verbose = 0
)

## S3 method for class 'kmsurmiserelation'
plot(
  x,
  ...,
  horizontal = FALSE,
  colors = NULL,
  keepNames = TRUE,
  vertexshape = "circle",
  arrowhead = "none",
  arrowtail = "none",
  verbose = 0
)

## S3 method for class 'kmlearningpathmatrices'
plot(
  x,
  ...,
  horizontal = FALSE,
  colors = NULL,
  keepNames = TRUE,
  itemsep = ",",
  braces = TRUE,
  vertexshape = "oval",
  arrowhead = "none",
  arrowtail = "none",
  edgelabel = FALSE,
  verbose = 0,
  structure = NULL,
  pathcolors = c("red", "blue", "gold", "deeppink", "brown", "violet", "green")
)

## S3 method for class 'kmlearningpathmatrix'
plot(
  x,
```

```
  ...,
  horizontal = FALSE,
  colors = NULL,
  keepNames = TRUE,
  itemsep = ",",
  braces = TRUE,
  vertexshape = "oval",
  arrowhead = "none",
  arrowtail = "none",
  edgelabel = FALSE,
  verbose = 0,
  structure = NULL,
  pathcolor = "blue"
)

## S3 method for class 'kmsurmisefunction'
plot(
  x,
  ...,
  space = FALSE,
  horizontal = FALSE,
  colors = "Beige",
  keepNames = TRUE,
  itemsep = ",",
  braces = TRUE,
  vertexshape = "oval",
  arrowhead = "none",
  arrowtail = "none",
  edgelabel = FALSE,
  verbose = 0,
  highlightcolor = "red"
)
```

Arguments

x	Binary matrix representing a family of sets
...	Optional inherited parameters
state	Knowledge state whose neighbourhood is to be pictured (Default 'NULL") If state is NULL, the neighbourhood is depicted as a 'normal' family of sets (kmfamset) and a respective warning is issued.
horizontal	Boolean defining orientation of the graph, default FALSE
colors	Color value or vector (default NULL).
keepNames	Keep item names (default TRUE)
itemsep	Item separator in sets (default ','; only for families of states)
braces	Put braces around vertices (default TRUE; only for families of states)
vertexshape	Shape of the vertex objects, e.g. circle, oval, box, or none. See Graphviz Node Shapes for a complete list of possible values.

arrowhead	Form of the arrow head, e.g. vee or none (default). See Graphviz Arrow Types for a complete list of possible values. This may be used for vertical graphs although none is the standard there.
arrowtail	Form of the arrow tail, e.g. vee or none (default). See Graphviz Arrow Types for a complete list of possible values. This should be used for horizontal graphs.
edgelabel	Boolean whether to label the edges of the diagram (default FALSE)
verbose	Verbosity level (0 (default), 1, or 2)
structure	Knowledge structure within which the path shall be depicted
pathcolors	Colors for the individual learning paths
pathcolor	Colors for the individual learning paths (default: blue)
space	Should basis or space de depicted? (default FALSE)
highlightcolor	Color for highlighting

Details

plot takes a matrix representing a family of sets (knowledge states) or a surmise relation and a color vector, and draws a Hasse diagram. If the color vector is NULL the states are drawn in green, the items in the relation are drawn in orange.

For a surmise relation, if there are equivalent items, they are contracted into one vertex labelled 'a ~ b ~ ...' for equivalent items a, b, ...

If the plot is to be used within a Shiny app, it must be included with grVizOutput() and renderGrViz() from the DiagrammeR package (plotOutput() and renderPlot() do not work).

Please note that, for equivalent items in a relation plot, the prerequisites are lost.

If the KST object to be plotted is a family of sets (kmfamset) **and** a state is specified, the colors vector must have four elements, (i) the color for the state, (ii) the color for its upper neighbours, (iii) the color for its lower neighbours, and (iv) the color for all other states.

See Also

Other Functions for plotting and printing knowledge structures: kmcolors(), kmprettyprint()

Examples

```
## Not run:
plot(phsg$basis)

sp <- kmunionclosure(phsg$basis)
n <- kmneighbourhood(phsg$basis[3,], sp, include=TRUE)
plot(n, state=phsg$basis[3,])

m <- matrix(c(1,0,0,1,1,1,1,1,1), ncol=3, byrow=FALSE)
class(m) <- unique(c("kmsurmiserelation", class(m)))
plot(m, vertexshape="oval")

## End(Not run)
```

readwrite *Knowledge spaces on reading and writing abilities*

Description

Bases of knowledge spaces on reading/writing abilities obtained from querying experts.

Usage

```
readwrite
```

Format

A list containing four bases (rw1 to rw3, and rwmaj) in binary matrix form. Each matrix has 48 columns representing the different knowledge items and a varying number of rows containing the basis elements.

Details

Three experts were queried about prerequisite relationships between 48 items on reading and writing abilities (Dowling, 1991; 1993). A forth basis represents those prerequisite relationships on which the majority of the experts agree (Dowling & Hockemeyer, 1998).

References

Dowling, C. E. (1991). *Constructing Knowledge Structures from the Judgements of Experts.* Habilitationsschrift, Technische Universität Carolo-Wilhelmina, Braunschweig, Germany.

Dowling, C. E. (1993). Applying the basis of a knowledge space for controlling the questioning of an expert. *Journal of Mathematical Psychology, 37,* 21–48.

Dowling, C. E. & Hockemeyer, C. (1998). Computing the intersection of knowledge spaces using only their basis. In Cornelia E. Dowling, Fred S. Roberts, & Peter Theuns, editors, *Recent Progress in Mathematical Psychology,* pp. 133–141. Lawrence Erlbaum Associates Ltd., Mahwah, NJ.

xpl *Small example knowledge space*

Description

Basis and space matrix, surmise relation and surmise function of a small fictional knowledge space, and two data sets (data (7 patterns) and sim (500 patterns)to be used in examples. The latter was produced from the space with `kmsimulate()` with beta and eta values of 0.1.

Usage

```
xpl
```

Format

A list containing the basis, the space, the surmise relation, the surmise function, and the two data matrices data and sim.

Index

∗ **Closure operators**
kmclosure, 16
kmintersectionclosure, 27
kmunionclosure, 49
∗ **Constructors**
kmfamset, 21
kmqspace, 36
kmspace, 40
kmstructure, 43
∗ **Deprecated functions and methods**
kmbasisdiagram, 14
kmhasse, 24
kmSF2basis, 39
kmSR2basis, 41
kmSRdiagram, 42
∗ **Different representations for knowledge structures**
kmbasis, 13
kmspace, 40
kmsurmisefunction, 45
kmsurmiserelation, 46
∗ **Functions for fringes & neighbourhoods**
kmbasisfringe, 15
kmbasisneighbourhood, 16
kmfringe, 21
kmneighbourhood, 31
∗ **Functions for plotting and printing knowledge structures**
kmcolors, 17
kmprettyprint, 33
plot, 51
∗ **Functions for wellgradedness & learning paths**
kmgradations, 23
kmiswellgraded, 27
kmlearningpathmatrices, 28
kmlearningpaths, 29
∗ **Functions for working with data**
kmdist, 18
kmgenerate, 22
kmiita2SR, 25
kmsimulate, 39
kmSRvalidate, 42
kmvalidate, 50
∗ **Functions for working with knowledge structures**
kmeqreduction, 19
kmexpand, 20
kmintersection, 26
kmmesh, 30
kmnotions, 32
kmsubstructure, 44
kmtrivial, 47
kmunion, 48
∗ **Knowledge assessment functions**
kmassess, 5
kmassessbayesian, 8
kmassesshalfsplit, 9
kmassessinformative, 10
kmassessmentsimulation, 11
kmassessmultiplicative, 12
∗ **Properties of knowledge structures**
kmheights, 24
kmiswellgraded, 27
kmnotions, 32
∗ **Utilities**
binary_matrix_product, 3
kmdoubleequal, 19
kmminimalfamset, 30
kmsetiselement, 38
kmsymmsetdiff, 46
kmtrivial, 47
∗ **datasets**
cad, 4
fractions, 4
phsg, 51
readwrite, 55
xpl, 55

Assessment *(*kmassess*)*, 5

binary_matrix_product, 3
binary_matrix_product(), *19*, *31*, *38*, *47, 48*

cad, 4

fractions, 4

kmassess, 5
kmassess(), *9–13*
kmassessbayesian, 8
kmassessbayesian(), *8*, *10–13*
kmassesshalfsplit, 9
kmassesshalfsplit(), *8, 9*, *11–13*
kmassessinformative, 10
kmassessinformative(), *8–10*, *12, 13*
kmassessmentsimulation, 11
kmassessmentsimulation(), *8–11*, *13*
kmassessmultiplicative, 12
kmassessmultiplicative(), *8–12*
kmbasis, 13
kmbasis(), *41*, *45, 46*
kmbasisdiagram, 14
kmbasisdiagram(), *24*, *39*, *41, 42*
kmbasisfringe, 15
kmbasisfringe(), *16*, *22*, *32*
kmbasisinnerfringe *(*kmbasisfringe*)*, 15
kmbasisneighbourhood, 16
kmbasisneighbourhood(), *15*, *22*, *32*
kmbasisouterfringe *(*kmbasisfringe*)*, 15
kmclosure, 16
kmclosure(), *27*, *49*
kmcolors, 17
kmcolors(), *36*, *54*
kmdist, 18
kmdist(), *22*, *25*, *40*, *43*, *50*
kmdoubleequal, 19
kmdoubleequal(), *3*, *31*, *38*, *47, 48*
kmeqreduction, 19
kmeqreduction(), *20*, *26*, *30*, *32*, *44*, *48, 49*
kmexpand, 20
kmexpand(), *20*, *26*, *30*, *32*, *44*, *48, 49*
kmfamset, 21
kmfamset(), *37*, *41*, *43*
kmfringe, 21
kmfringe(), *15, 16*, *32*
kmgenerate, 22
kmgenerate(), *18*, *25*, *40*, *43*, *50*
kmgradations, 23
kmgradations(), *28, 29*
kmhasse, 24
kmhasse(), *15*, *39*, *41, 42*
kmheights, 24
kmheights(), *28*, *32*
kmiita2SR, 25
kmiita2SR(), *18*, *22*, *40*, *43*, *50*
kminnerfringe *(*kmfringe*)*, 21
kmintersection, 26
kmintersection(), *20*, *30*, *32*, *44*, *48, 49*
kmintersectionclosure, 27
kmintersectionclosure(), *17*, *49*
kmiswellgraded, 27
kmiswellgraded(), *23, 24*, *28, 29*, *32*
kmlearningpathmatrices, 28
kmlearningpathmatrices(), *23*, *28, 29*
kmlearningpaths, 29
kmlearningpaths(), *23*, *28*
kmmaximalspace *(*kmtrivial*)*, 47
kmmesh, 30
kmmesh(), *20*, *26*, *32*, *44*, *48, 49*
kmminimalfamset, 30
kmminimalfamset(), *3*, *19*, *38*, *47, 48*
kmminimalspace *(*kmtrivial*)*, 47
kmneighbourhood, 31
kmneighbourhood(), *15, 16*, *22*
kmnneighbourhood *(*kmneighbourhood*)*, 31
kmnotions, 32
kmnotions(), *20*, *24*, *26*, *28*, *30*, *44*, *48, 49*
kmouterfringe *(*kmfringe*)*, 21
kmprettyprint, 33
kmprettyprint(), *18*, *54*
kmqspace, 36
kmqspace(), *21*, *41*, *43*
kmrefine, 37
kmsassess *(*kmassess*)*, 5
kmsetdistance *(*kmsymmsetdiff*)*, 46
kmsetiselement, 38
kmsetiselement(), *3*, *19*, *31*, *47, 48*
kmSF2basis, 39
kmSF2basis(), *15*, *24*, *41, 42*
kmsimulate, 39
kmsimulate(), *18*, *22*, *25*, *43*, *50*
kmspace, 40
kmspace(), *14*, *21*, *37*, *43*, *45, 46*
kmSR2basis, 41

kmSR2basis(), *15*, *24*, *39*, *42*
kmSRdiagram, 42
kmSRdiagram(), *15*, *24*, *39*, *41*
kmSRvalidate, 42
kmSRvalidate(), *18*, *22*, *25*, *40*, *50*
kmstructure, 43
kmstructure(), *21*, *37*, *41*
kmsubstructure, 44
kmsubstructure(), *20*, *26*, *30*, *32*, *48, 49*
kmsurmisefunction, 45
kmsurmisefunction(), *14*, *41*, *46*
kmsurmiserelation, 46
kmsurmiserelation(), *14*, *41*, *45*
kmsymmsetdiff, 46
kmsymmsetdiff(), *3*, *19*, *31*, *38*, *48*
kmtrivial, *3*, *19, 20*, *26*, *30*–*32*, *38*, *44*, *47*, 47, *49*
kmunion, 48
kmunion(), *20*, *26*, *30*, *32*, *44*, *48*
kmunionclosure, 49
kmunionclosure(), *17*, *27*
kmvalidate, 50
kmvalidate(), *18*, *22*, *25*, *40*, *43*

phsg, 51
plot, *18*, *36*, 51

readwrite, 55

xpl, 55