Package 'eatATA'

December 7, 2020

Type Package

Title Create Constraints for Small Test Assembly Problems

Version 0.9.1

Description Provides simple functions to create constraints for small test assembly problems (e.g. van der Linden (2005, ISBN: 978-0-387-29054-6)) using sparse matrices. Currently, 'GLPK', 'lpSolve', and 'Gurobi' are supported as solvers. The 'gurobi' package is not available from any mainstream repository; see <https://www.gurobi.com/downloads/>.

License GPL

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports Matrix, Rglpk, lpSolve

RoxygenNote 7.1.1

Suggests testthat (>= 2.1.0), covr, knitr, rmarkdown, readxl

Enhances gurobi

VignetteBuilder knitr

NeedsCompilation no

Author Benjamin Becker [aut, cre],
Dries Debeer [aut]

Maintainer Benjamin Becker <b.becker@iqb.hu-berlin.de>

Repository CRAN

Date/Publication 2020-12-07 17:30:04 UTC

R topics documented:

analyzeBlockExclusion 2
analyzeComplexBlockExclusion 3
appendSolution 4
autoItemValuesMinMax 5
calculateExpectedRT 6

calculateIIF . . . 7
computeTargetValues . . . 8
depletePoolConstraint . . . 11
dummiesToFactor . . . 11
gurobiExample . . . 12
inspectSolution . . . 12
itemCategoryConstraint . . . 13
itemCategoryRange . . . 14
itemExclusionConstraint . . . 16
itemExclusionTuples . . . 17
items . . . 18
itemsPerFormConstraint . . . 19
itemTargetConstraint . . . 20
itemUsageConstraint . . . 20
itemValuesConstraint . . . 21
itemValuesRange . . . 22
matrixExclusionTuples . . . 24
useSolver . . . 25

Index **27**

`analyzeBlockExclusion` *Analyze block exclusiveness*

Description

Use exclusion tuples information to determine which assembled test blocks are exclusive.

Usage

```
analyzeBlockExclusion(solverOut, items, idCol, exclusionTuples)
```

Arguments

`solverOut`	Object created by `useSolver`.
`items`	Original `data.frame` containing information on item level.
`idCol`	Column name with item IDs in the `items` `data.frames`.
`exclusionTuples`	`data.frame` with two columns, containing tuples with item IDs which should be in test forms exclusively. Must be the same object as used in `itemExclusionConstraint`.

Details

If exclusion tuples have been used to assemble test forms (using the `itemExclusionConstraint` function), the resulting item blocks might also be exclusive. Using the initially used item exclusion tuples and the optimal solution given by `useSolver` this function determines, which item blocks are exclusive and can not be together in an assembled test form.

Value

A `data.frame` of block exclusions.

Examples

```
## Full workflow using itemExclusionTuples
# Example data.frame
items <- data.frame(ID = c("items1", "items2", "items3", "items4"),
                    exclusions = c("items2, items3", NA, NA, NA),
                    stringsAsFactors = FALSE)

# Create tuples
exTuples2 <- itemExclusionTuples(items = items, idCol = "ID", exclusions = "exclusions",
                   sepPattern = ", ")

#' ## Create constraints
exclusion_constraint <- itemExclusionConstraint(nForms = 2, exclusionTuples = exTuples2,
                                                itemIDs = items$ID)
depletion_constraint <- depletePoolConstraint(2, nItems = 4)
target_constraint <- itemTargetConstraint(nForms = 2, nItems = 4,
                                          itemValues = c(3, 1.5, 2, 4), targetValue = 1)

opt_solution <- useSolver(list(exclusion_constraint, target_constraint,
                                       depletion_constraint),
                                  nForms = 2, nItems = 4, itemIDs = items$ID)

analyzeBlockExclusion(opt_solution, items = items, idCol = "ID",
                      exclusionTuples = exTuples2)
```

`analyzeComplexBlockExclusion`
Analyze complex block exclusiveness

Description

Use exclusion tuples information from independent test assembly problems to determine which assembled test blocks are exclusive.

Usage

```
analyzeComplexBlockExclusion(
  solverOut_list,
  items_list,
  idCol,
  exclusionTuples_list
)
```

Arguments

solverOut_list	List of objects created by useSolver.
items_list	List of original data.frame containing information on item level.
idCol	Column name with item IDs in the items data.frames.
exclusionTuples_list	List of data.frames with two columns, containing tuples with item IDs which should be in test forms exclusively. Must be the same objects as used in itemExclusionConstraint.

Details

If exclusion tuples have been used to assemble test forms (using the itemExclusionConstraint function), the resulting item blocks might also be exclusive. Using the initially used item exclusion tuples and the optimal solution given by useSolver this function determines, which item blocks are exclusive and can not be together in an assembled test form. analyzeComplexBlockExclusion allows analyzing block exclusiveness from separate test assembly problems. This can be useful if test forms consist of blocks containing different domains or dimensions.

Value

A data.frame of block exclusions.

Examples

```
## Full workflow using itemExclusionTuples
# tbd
```

---

appendSolution *Append a* useSolver *output*

Description

Append a useSolver output of a successfully solved optimization problem to the initial item pool data.frame.

Usage

```
appendSolution(solverOut, items, idCol)
```

Arguments

solverOut	Object created by useSolver function.
items	Original data.frame containing information on item level.
idCol	Column name in items containing item IDs. These will be used for matching to the solver output.

Details

This function merges the initial item pool information in `items` to the solver output in `solverOut`.

Value

A `data.frame`.

Examples

```
## Example item pool
items <- data.frame(ID = 1:10,
itemValues = c(-4, -4, -2, -2, -1, -1, 20, 20, 0, 0))

## Test Assembly
usage <- itemUsageConstraint(nForms = 2, nItems = 10, operator = "=", targetValue = 1)
perForm <- itemsPerFormConstraint(nForms = 2, nItems = 10, operator = "=", targetValue = 5)
target <- itemTargetConstraint(nForms = 2, nItems = 10,
                               itemValues = items$itemValues,
                               targetValue = 0)
sol <- useSolver(allConstraints = list(usage, perForm, target),
                                  nForms = 2, itemIDs = items$ID, solver = "lpSolve")

## Append Solution to existing item information
out <- appendSolution(sol, items = items, idCol = "ID")
```

`autoItemValuesMinMax` *Create single value constraints with minimum and maximum.*

Description

`itemValuesDeviation` creates constraints related to an item parameter/value. `autoItemValuesMixMax` automatically determines the appropriate `targetValue` and then calls `itemValuesDeviation`. The function only works for (dichotomous) dummy indicators with values 0 and 1.

Usage

```
autoItemValuesMinMax(
  nForms,
  itemValues,
  allowedDeviation = NULL,
  relative = FALSE,
  verbose = TRUE
)
```

Arguments

nForms	Number of forms to be created.
itemValues	Item parameter/values for which the sum per test form should be constrained.
allowedDeviation	Numeric value of length 1. How much deviance is allowed from target values?
relative	Is the allowedDeviation expressed as a proportion?
verbose	Should calculated values be reported?

Details

Two scenarios are possible when automatically determining the target value: (a) Either items with the selected property could be exactly distributed across test forms or (b) this is not possible. An example would be 2 test forms and 4 multiple choice items (a) or 2 test forms and 5 multiple choice items (b). If (a), the tolerance level works exactly as one would expect. If (b) the tolerance level is adapted, meaning that if tolerance level is 0 in example (b), allowed values are 2 or 3 multiple choice items per test form.

Value

A sparse matrix.

Examples

```
autoItemValuesMinMax(2, itemValues = c(0, 1, 0, 1))
```

---

calculateExpectedRT *Calculate Expected Response Times*

Description

Calculate expected response times given item parameters of the log normal response time model.

Usage

```
calculateExpectedRT(lambda, phi = rep(1, length(lambda)), zeta, sdEpsi)
```

Arguments

lambda	Vector of time intensity parameters.
phi	[optional] Vector of time sensitivity parameters.
zeta	Vector of person speed parameters.
sdEpsi	Vector of item specific residual variances.

Details

Expected response times are calculated according to the log normal response time model by van der Linden (2006) or Klein Entink et al. (2009). If `phi` is 1, the model by van der Linden (2006) is used. Either a single set of parameters of vectors of each parameters can be supplied.

The calculation is based on Fenton (1960). For the model by van der Linden (2006), the calculation was first introduced by van der Linden (2011).

Value

a matrix, with columns for different `zeta` and rows for different items

References

Fenton, L. (1960). The sum of log-normal probability distributions in scatter transmission systems. *IRE Transactions on Communication Systems*, 8, 57-67.

Klein Entink, R. H., Fox, J.-P., & van der Linden, W. J. (2009). A multivariate multilevel approach to the modeling of accuracy and speed of test takers. *Psychometrika*, 74(1), 21-48.

van der Linden, W. J. (2006). A lognormal model for response times on test items. *Journal of Educational and Behavioral Statistics, 31(2), 181-204*.

van der Linden, W. J. (2011). Test design and speededness. *Journal of Educational Measurement*, 48(1), 44-60.

Examples

```
# expected RT for a single item (van der Linden model)
calculateExpectedRT(lambda = 3.8, zeta = 0, sdEpsi = 0.3)

# expected RT for multiple items (van der Linden model)
calculateExpectedRT(lambda = c(4.1, 3.8, 3.5), zeta = 0,
                    sdEpsi = c(0.3, 0.4, 0.2))

# TIF for multiple items and multiple ability levels (1PL model)
calculateExpectedRT(lambda = c(3.7, 4.1, 3.8), phi = c(1.1, 0.8, 0.5),
                     zeta = c(-1, 0, 1), sdEpsi = c(0.3, 0.4, 0.2))
```

---

calculateIIF *Calculate Item Information Function*

Description

Calculate item information function given item parameters of the 1PL, 2PL or 3PL IRT model.

Usage

```
calculateIIF(A = rep(1, length(B)), B, C = rep(0, length(B)), theta)
```

Arguments

A	Vector of discrimination parameters.
B	Vector of difficulty parameters.
C	Vector of guessing parameters.
theta	Vector of time intensity parameters.

Value

a matrix, with columns for different theta and rows for different items

References

van der Linden, W. J. (2005). *Linear models for optimal test design*. New York, NY: Springer.

Examples

```
# TIF for a single item (2PL model)
calculateIIF(A = 0.8, B = 1.1, theta = 0)

# TIF for multiple items (1PL model)
calculateIIF(B = c(1.1, 0.8, 0.5), theta = 0)

# TIF for multiple items and multiple ability levels (1PL model)
calculateIIF(A = c(0.7, 1.1, 0.8), B = c(1.1, 0.8, 0.5),
             theta = c(-1, 0, 1))
```

---

computeTargetValues *Compute target values based on the item pool.*

Description

Compute target values for item values/categories based on the number of items in the item pool, the number of test forms to assemble and the number of items in each test form (i.e., test length).

Usage

```
computeTargetValues(
  itemValues,
  nForms,
  testLength = NULL,
  allowedDeviation = NULL,
  relative = FALSE
)

## Default S3 method:
computeTargetValues(
```

```
  itemValues,
  nForms,
  testLength = NULL,
  allowedDeviation = NULL,
  relative = FALSE
)

## S3 method for class 'factor'
computeTargetValues(
  itemValues,
  nForms,
  testLength = NULL,
  allowedDeviation = NULL,
  relative = FALSE
)
```

Arguments

itemValues	Item parameter/values for which the sum per test form should be constrained.
nForms	Number of forms to be created.
testLength	to be documented.
allowedDeviation	Numeric value of length 1. How much deviance is allowed from target values?
relative	Is the allowedDeviation expressed as a proportion?

Details

Both for numerical and categorical item values, the target values are the item pool average scaled by the ratio of items in the forms and items in the item pool. The behavior of the function changes depending on the class of itemValues.

When itemValues is a numerical vector, an when allowedDeviation is NULL (the default), only one target value is computed. This value could be used in the targetConstraint-function. Otherwise (i.e., allowedDeviation is a numerical value), the target is computed, but a minimal and a maximal (target)value are returned, based on the allowed deviation. When relative == TRUE the allowed deviation should be expressed as a proportion. In that case the minimal and maximal values are a computed proportionally.

When itemValues is a factor, it is assumed that the item values are item categories, and hence only whole valued frequencies are returned. To be more precise, a matrix with the minimal and maximal target frequencies for every level of the factor are returned. When allowedDeviation is NULL, the difference between the minimal and maximal value is one (or zero). As a consequence, dummy-item values are best specified as a factor (see examples).

Value

a vector or a matrix with target values (see details)

Methods (by class)

- `default`: compute target values
- `factor`: compute target frequencies for item categories

Examples

```
## Assume an item pool with 50 items with random item information values (iif) for
## a given ability value.
set.seed(50)
itemInformations <- runif(50, 0.5, 3)

## The target value for the test information value (i.e., sum of the item
## informations) when three test forms of 10 items are assembled is:
computeTargetValues(itemInformations, nForms = 3, testLength = 10)

## The minimum and maximum test iformation values for an allowed deviation of
## 10 percent are:
computeTargetValues(itemInformations, nForms = 3, allowedDeviation = .10,
   relative = TRUE, testLength = 10)


## items$MC is dummy variable indication which items in the pool are multiple choise
str(items$MC)

## when used as a numerical vector, the dummy is not treated as a categorical
## indicator, but rather as a numerical value.
computeTargetValues(items$MC, nForms = 14)
computeTargetValues(items$MC, nForms = 14, allowedDeviation = 1)

## Therefore, it is best to convert dummy variables into a factor, so that
## automatically freqyencies are returned
MC_factor <- factor(items$MC, labels = c("not MC", "MC"))
computeTargetValues(MC_factor, nForms = 14)
computeTargetValues(MC_factor, nForms = 3)

## The computed minimum and maximum frequencies can be used to create contstraints.
MC_ranges <- computeTargetValues(MC_factor, nForms = 3)
itemCategoryRange(3, nItems = length(MC_factor), MC_factor, range = MC_ranges)

## When desired, the automatically computed range can be adjusted by hand. This
##  can be of use when only a limited set of the categories should be constrained.
##  For instance, when only the multiple-choice items should be constrained, and
##  the non-multiple-choice items should not be constrained, the minimum and
##  maximum value can be set to a very small and a very high value, respectively.
##  Or to other sensible values.
MC_ranges["not MC", ] <- c(0, 40)
MC_ranges
itemCategoryRange(3, nItems = length(MC_factor), MC_factor, range = MC_ranges)
```

depletePoolConstraint *Use complete item pool.*

Description

Creates constraints that assure that every item in the item pool is used (at least) once. Essentially a wrapper around `itemUsageConstraint`.

Usage

```
depletePoolConstraint(nForms, nItems)
```

Arguments

`nForms`	Number of forms to be created.
`nItems`	Number of items in the item pool.

Value

A sparse matrix.

Examples

```
depletePoolConstraint(2, 10)
```

dummiesToFactor *Convert dummy variables to factor.*

Description

Convert multiple dummy variables into a single factor variable.

Usage

```
dummiesToFactor(dat, dummies, facVar, nameEmptyCategory = "_none_")
```

Arguments

`dat`	A `data.frame`.
`dummies`	Character vector containing the names of the dummy variables in the `data.frame`.
`facVar`	Name of the factor variable, that should be created.
`nameEmptyCategory`	a character of length 1 that defines the name of cases for which no dummy is equal to one.

Details

The content of a single factor variable can alternatively be stored in multiple dichotomous dummy variables coded with 0/1 or NA/1. 1 always has to refer to "this category applies". The function requires factor levels to be exclusive (i.e. only one factor level applies per row.).

Value

A data.frame containing the newly created factor.

Examples

```
# Example data set
tdat <- data.frame(ID = 1:3, d1=c(1, 0, 0), d2 = c(0, 1, 0), d3 = c(0, 0, 1))

dummiesToFactor(tdat, dummies = c("d1", "d2", "d3"), facVar = "newFac")
```

---

gurobiExample *Example* Gurobi *output.*

Description

An example Gurobi output containing the optimal solution for the example in the vignette.

Usage

```
gurobiExample
```

Format

A list.

---

inspectSolution *Inspect a* useSolver *output*

Description

Process a useSolver output of a successfully solved optimization problem to a list so it becomes humanly readable.

Usage

```
inspectSolution(solverOut, items, idCol, colNames, colSums = TRUE)
```

Arguments

solverOut	Object created by useSolver function.
items	Original data.frame containing information on item level.
idCol	Column name in items containing item IDs. These will be used for matching to the solver output.
colNames	Which columns should be used from the items data.frame?
colSums	Should column sums be calculated in the output? Only works if all columns are numeric.

Details

This function merges the initial item pool information in items to the solver output in solverOut. Relevant columns can be selected via colNames. Column sums within test forms are calculated if possible and if colSum is set to TRUE.

Value

A list with assembled blocks as entries. Rows are the individual items. A final row is added, containing the sums of each column.

Examples

```
## Example item pool
items <- data.frame(ID = 1:10,
itemValues = c(-4, -4, -2, -2, -1, -1, 20, 20, 0, 0))

## Test Assembly
usage <- itemUsageConstraint(nForms = 2, nItems = 10, operator = "=", targetValue = 1)
perForm <- itemsPerFormConstraint(nForms = 2, nItems = 10, operator = "=", targetValue = 5)
target <- itemTargetConstraint(nForms = 2, nItems = 10,
                               itemValues = items$itemValues,
                               targetValue = 0)
sol <- useSolver(allConstraints = list(usage, perForm, target),
                                  nForms = 2, itemIDs = items$ID, solver = "lpSolve")

## Inspect Solution
out <- inspectSolution(sol, items = items, idCol = "ID", colNames = "itemValues")
```

---

itemCategoryConstraint

Create item category constraints.

Description

Create constraints related to item categories/groupings (as represented by itemCategories). That is, the created constraints assure that the number of items of each category per test form is either (a) smaller or equal than (operator = "<="), (b) equal to (operator = "="), or (c) greater than or equal to (operator = ">=") the corresponding targetValues.

Usage

```
itemCategoryConstraint(
  nForms,
  nItems,
  itemCategories,
  operator = c("<=", "=", ">="),
  targetValues
)
```

Arguments

nForms	Number of forms to be created.
nItems	Number of items in the item pool.
itemCategories	a factor representing the categories/grouping of the items
operator	a character indicating which operator should be used in the constraints, with three possible values: "<=", "=", or ">=". See details for more information.
targetValues	an integer vector representing the target number per category. The order of the target values should correspond with the order of the levels of the factor in itemCategory.

Value

A sparse matrix.

Examples

```
## constraints to make sure that there are at least 3 items of each item type
## in each test form
nItems <- 30
item_type <- factor(sample(1:3, size = nItems, replace = TRUE))
itemCategoryConstraint(2, nItems, item_type, ">=", targetValues = c(3, 3, 3))
```

---

itemCategoryRange *Create item category constraints with minimum and maximum.*

Description

itemCategoriesRange, itemCategoriesMin, and itemCategoriesMax create constraints related to item categories/groupings (as represented by itemCategories). That is, the created constraints assure that the number of items of each category per test form is either smaller or equal than the specified max, greater than or equal to min or both range.

Usage

```
itemCategoryRange(nForms, nItems, itemCategories, range)

itemCategoryMin(nForms, nItems, itemCategories, min)

itemCategoryMax(nForms, nItems, itemCategories, max)

itemCategoryDeviation(
  nForms,
  nItems,
  itemCategories,
  targetValues,
  allowedDeviation,
  relative = FALSE
)
```

Arguments

nForms	Number of forms to be created.
nItems	Number of items in the item pool.
itemCategories	a factor representing the categories/grouping of the items
range	a matrix with two columns representing the the minimal and the maximum frequency of the items from each level/category itemCategories
min	the minimal sum of the itemValues per test form
max	the minimal sum of the itemValues per test form
targetValues	an integer vector representing the target number per category. The order of the target values should correspond with the order of the levels of the factor in itemCategory.
allowedDeviation	the maximum allowed deviation from the targetValue
relative	a logical expressing whether or not the allowedDeviation should be interpreted as a proportion of the targetValue

Details

itemCategoriesDeviation also constrains the minimal and the maximal value of the number of items of each category per test form, but based on chosen targetValues, and maximal allowed deviations (i.e., allowedDeviation) from those targetValues.

Value

A sparse matrix.

Functions

- `itemCategoryMin`: constrain minimum value
- `itemCategoryMax`: constrain maximum value
- `itemCategoryDeviation`: constrain the distance form the `targetValues`

Examples

```
## constraints to make sure that there are at least 2 and maximally 4
##  items of each item type in each test form
nItems <- 30
item_type <- factor(sample(1:3, size = nItems, replace = TRUE))
itemCategoryRange(2, nItems, item_type, range = cbind(min = rep(2, 3), max = rep(4, 3)))

## or alternatively
itemCategoryDeviation(2, nItems, item_type, targetValues = rep(3, 3), allowedDeviation = rep(4, 3))
```

`itemExclusionConstraint` *Create item exclusion constraints.*

Description

Create constraints that prohibit that item pairs occur in the same test forms.

Usage

```
itemExclusionConstraint(nForms, exclusionTuples, itemIDs)
```

Arguments

`nForms`	Number of forms to be created.
`exclusionTuples`	`data.frame` with two columns, containing tuples with item IDs which should be in test forms exclusively.
`itemIDs`	Character vector of item IDs in correct ordering.

Details

Item exclusion pairs can, for example, be created by the function `itemExclusionTuples`.

Value

A sparse matrix.

Examples

```
## item-IDs
IDs <- c("item1", "item2", "item3", "item4")

## tuples: Item 1 can not be in the test form as item 2 and 3
exTuples <- data.frame(v1 = c("item1", "item1"), v2 = c("item2", "item3"),
                       stringsAsFactors = FALSE)

## Create constraints
itemExclusionConstraint(nForms = 2, exclusionTuples = exTuples, itemIDs = IDs)


########
## Full workflow using itemExclusionTuples
# Example data.frame
items <- data.frame(ID = c("item1", "item2", "item3", "item4"),
                    exclusions = c("item2, item3", NA, NA, NA))

# Create tuples
exTuples2 <- itemExclusionTuples(items = items, idCol = "ID", exclusions = "exclusions",
                   sepPattern = ", ")

#' ## Create constraints
itemExclusionConstraint(nForms = 2, exclusionTuples = exTuples2, itemIDs = IDs)
```

`itemExclusionTuples` *Create item exclusion tuples.*

Description

If item exclusions are stored as a character vector, `itemExclusionTuples` separates this vector and creates item pairs ('tuples').

Usage

```
itemExclusionTuples(items, idCol = "ID", exclusions, sepPattern = ", ")
```

Arguments

`items`	A `data.frame` with information on an item pool.
`idCol`	Name of the item ID column.
`exclusions`	Name of the exclusion column.
`sepPattern`	String which should be used for separating item IDs in the `exclusions` column..

Details

Exclusion tuples can be used by `itemExclusionConstraint` to set up exclusion constraints. Note that a separator pattern has to be used consistently throughout the column (e.g. `", "`).

Value

A `data.frame` with two columns.

Examples

```
# Example data.frame
items <- data.frame(ID = c("items1", "items2", "items3", "items4"),
                    exclusions = c("items2, items3", NA, NA, NA))

# Create tuples
itemExclusionTuples(items = items, idCol = "ID", exclusions = "exclusions",
                    sepPattern = ", ")
```

---

items *Small artificial item pool example.*

Description

A `data.frame` containing 80 items with different categorical and metric properties.

Usage

```
items
```

Format

A `data.frame` .

Item_ID Item identifier.

exclusions Items which can not be in the same test form.

RT_in_min Average response times in minutes. `2.5` equals 2 minutes and 30 seconds, for example.

subitems Number of sub items.

MC, CMC, short_answer, open Answer formats.

diff_1, diff_2, diff_3, diff_4, diff5 Difficulty categories.

`itemsPerFormConstraint` *Create number of items per test form constraints.*

Description

Creates constraints related to the number of items in each test form.

Usage

```
itemsPerFormConstraint(
  nForms,
  nItems,
  operator = c("<=", "=", ">="),
  targetValue
)
```

Arguments

`nForms`	Number of forms to be created.
`nItems`	Number of items in the item pool.
`operator`	a character indicating which operator should be used in the constraints, with three possible values: `"<="`, `"="`, or `">="`. See details for more information.
`targetValue`	The target value to be used in the constraints. That is, the number of items per form.

Details

The number of items per test form is constrained to be either (a) smaller or equal than (`operator = "<="`), (b) equal to (`operator = "="`), or (c) greater or equal than (`operator = ">="`) the chosen `value`.

Value

A sparse matrix.

Examples

```
## Constrain the test forms to have exactly five items
itemsPerFormConstraint(3, 20, operator = "=", targetValue = 5)
```

itemTargetConstraint *Create optimization constraints.*

Description

Create constraints that define the optimization goal of a automated test assembly problem.

Usage

```
itemTargetConstraint(nForms, nItems, itemValues, targetValue)
```

Arguments

nForms	Number of forms to be created.
nItems	Number of items in the item pool.
itemValues	Item parameter/values for which the sum per test form should be constrained
targetValue	The target value to be used in the constraints.

Value

A sparse matrix.

Examples

```
itemTargetConstraint(nForms = 2, nItems = 4, c(1, 0.5, 1.5, 2), targetValue = 1)
```

itemUsageConstraint *Create item usage constraints.*

Description

Creates constraints related to item usage. That is, the number of times an item is selected is constrained to be either (a) smaller or equal than (`operator = "<="`), (b) equal to (`operator = "="`), or (c) greater or equal than (`operator = ">="`) the chosen `value`.

Usage

```
itemUsageConstraint(
  nForms,
  nItems,
  operator = c("<=", "=", ">="),
  targetValue = 1
)
```

Arguments

nForms	Number of forms to be created.
nItems	Number of items in the item pool.
operator	a character indicating which operator should be used in the constraints, with three possible values: "<=", "=", or ">=". See details for more information.
targetValue	The value to be used in the constraints

Details

When operator = "<=" and value = 1 (the default), each item can be selected maximally once, which corresponds with assuring that there is no item overlap between the forms. When operator = "=" and value = 1, each item is used exactly once, which corresponds to no item-overlap and complete item pool depletion.

Value

A sparse matrix.

Examples

```
## create no-item overlap constraints with item pool depletion
##  for 2 test forms with an item pool of 20 items
itemUsageConstraint(2, 20, operator = "=", targetValue = 1)
```

---

itemValuesConstraint *Create single value constraints.*

Description

Create constraints related to an item parameter/value. That is, the created constraints assure that the sum of the item values (itemValues) per test form is either (a) smaller than or equal to (operator = "<="), (b) equal to (operator = "="), or (c) greater than or equal to (operator = ">=") the chosen targetValue.

Usage

```
itemValuesConstraint(
  nForms,
  nItems,
  itemValues,
  operator = c("<=", "=", ">="),
  targetValue
)
```

Arguments

nForms	Number of forms to be created.
nItems	Number of items in the item pool.
itemValues	Item parameter/values for which the sum per test form should be constrained.
operator	A character indicating which operator should be used in the constraints, with three possible values: "<=", "=", or ">=". See details for more information.
targetValue	the target test form value

Value

A sparse matrix.

Examples

```
## constraints to make sure that the sum of the item values (1:10) is between
## 4 and 6
rbind(
  itemValuesConstraint(2, 10, 1:10, operator = ">=", targetValue = 4),
  itemValuesConstraint(2, 10, 1:10, operator = "<=", targetValue = 6))
```

---

itemValuesRange *Create single value constraints with minimum and maximum.*

Description

itemValuesRange, itemValuesMin, and itemValuesMax create constraints related to an item parameter/value. That is, the created constraints assure that the sum of the itemValues is smaller than or equal to max, greater than or equal to min, or both range.

Usage

```
itemValuesRange(nForms, nItems, itemValues, range)

itemValuesMin(nForms, nItems, itemValues, min)

itemValuesMax(nForms, nItems, itemValues, max)

itemValuesDeviation(
  nForms,
  nItems,
  itemValues,
  targetValue,
  allowedDeviation,
  relative = FALSE
)
```

Arguments

nForms	Number of forms to be created.
nItems	Number of items in the item pool.
itemValues	Item parameter/values for which the sum per test form should be constrained.
range	a vector with two values, the the minimal and the maximum sum of the itemValues per test form, respectively
min	the minimal sum of the itemValues per test form
max	the minimal sum of the itemValues per test form
targetValue	the target test form value
allowedDeviation	the maximum allowed deviation from the targetValue
relative	a logical expressing whether or not the allowedDeviation should be interpreted as a proportion of the targetValue

Details

itemValuesDeviation also constrains the minimal and the maximal value of the sum of the itemValues, but based on a chosen and a maximal allowed deviation (i.e., allowedDeviation) from that targetValue.

Value

A sparse matrix.

Functions

- itemValuesMin: constrain minimum value
- itemValuesMax: constrain maximum value
- itemValuesDeviation: constrain the distance form the targetValue

Examples

```
## constraints to make sure that the sum of the item values (1:10) is between
## 4 and 6
itemValuesRange (2, 10, 1:10, range(min = 4, max = 6))

## or alternatively
itemValuesDeviation (2, 10, 1:10, targetValue = 5, allowedDeviation = 1)
```

`matrixExclusionTuples` *Create item exclusion tuples from matrix.*

Description

If item exclusions are stored as a matrix, `matrixExclusionTuples` transforms this format into item pairs ('tuples'). Information on exclusions has to be coded as `1` (items are exclusive) and `0` (items are not exclusive).

Usage

```
matrixExclusionTuples(exclMatrix)
```

Arguments

`exclMatrix`	A `data.frame` or `matrix` with information on item exclusiveness.

Details

Exclusion tuples can be used by `itemExclusionConstraint` to set up exclusion constraints.

Value

A `data.frame` with two columns.

Examples

```
# Example data.frame
exclDF <- data.frame(c(0, 1, 0, 0),
                     c(1, 0, 0, 1),
                     c(0, 0, 0, 0),
                     c(0, 1, 0, 0))
rownames(exclDF) <- colnames(exclDF) <- paste0("item_", 1:4)

# Create tuples
matrixExclusionTuples(exclDF)
```

useSolver *Use a solver for a list of constraints.*

Description

Use a mathematical programming solver to solve a list for constrains.

Usage

```
useSolver(
  allConstraints,
  nForms,
  itemIDs,
  solver = c("GLPK", "lpSolve", "Gurobi"),
  timeLimit = Inf,
  modelSense = c("min", "max"),
  ...
)
```

Arguments

allConstraints	List of constraints.
nForms	Number of forms to be created.
itemIDs	Character vector of item IDs.
solver	A character string indicating the solver to use.
timeLimit	The maximal runtime in seconds.
modelSense	A character string indicating whether to minimize or maximize the objective function.
...	Additional arguments for the solver.

Details

Wrapper around the functions of different solvers (gurobi::gurobi(),lpSolve::lp(),... for a list of constraints set up via eatATA. Rglpk is used per default.

Additional arguments can be passed through ... and vary from solver to solver (see their respective help pages, lp or Rglpk_solve_LP); for example time limits can not be set for lpSolve.

Value

A list with the following elements:

solution The object returned by the solver.

solver_function The solver function that was called, and can be called with the objects_for_solver.

objects_for_solver A list with the objects that are used in the call.

Examples

```
nForms <- 2
nItems <- 4

# create constraits
target <- itemTargetConstraint(nForms = nForms, nItems = nItems,
  c(1, 0.5, 1.5, 2), targetValue = 2)
noItemOverlap <- itemUsageConstraint(nForms, nItems = nItems, operator = "=")
testLength <- itemsPerFormConstraint(nForms = nForms, nItems = nItems,
  operator = "<=", 2)

# use a solver
result <- useSolver(list(target, noItemOverlap, testLength),
  nForms = nForms, itemIDs = paste0("Item_", 1:4),
  solver = "GLPK")
```

Index

∗ **datasets**
    gurobiExample, 12
    items, 18

analyzeBlockExclusion, 2
analyzeComplexBlockExclusion, 3
appendSolution, 4
autoItemValuesMinMax, 5

calculateExpectedRT, 6
calculateIIF, 7
computeTargetValues, 8

depletePoolConstraint, 11
dummiesToFactor, 11

gurobiExample, 12

inspectSolution, 12
itemCategoryConstraint, 13
itemCategoryDeviation *(*itemCategoryRange*)*, 14
itemCategoryMax *(*itemCategoryRange*)*, 14
itemCategoryMin *(*itemCategoryRange*)*, 14
itemCategoryRange, 14
itemExclusionConstraint, *2*, *4*, 16, *17*, *24*
itemExclusionTuples, *16*, 17
items, 18
itemsPerFormConstraint, 19
itemTargetConstraint, 20
itemUsageConstraint, 20
itemValuesConstraint, 21
itemValuesDeviation, *5*
itemValuesDeviation *(*itemValuesRange*)*, 22
itemValuesMax *(*itemValuesRange*)*, 22
itemValuesMin *(*itemValuesRange*)*, 22
itemValuesRange, 22

lp, *25*

matrixExclusionTuples, 24

Rglpk_solve_LP, *25*

useSolver, 25