Package ‘eatATA’

April 16, 2021

Type Package

Title Create Constraints for Small Test Assembly Problems

Version 0.11.1

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’, ‘Symphony’, and ‘Gurobi’ are supported as solvers. The ‘gurobi’ package is not available from any mainstream repository; see <https://www.gurobi.com/downloads/>.

License GPL

URL https://github.com/beckerbenj/eatATA

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, Rsymphony

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 2021-04-16 15:30:09 UTC
analyzeBlockExclusion

R topics documented:

analyzeBlockExclusion .................................................. 2
analyzeComplexBlockExclusion ....................................... 4
appendSolution ............................................................. 5
autoItemValuesMinMaxConstraint ..................................... 6
calculateExpectedRT ...................................................... 7
calculateIF ................................................................. 8
cappedMaximinObjective ................................................. 9
combineConstraints ....................................................... 10
computeTargetValues .................................................... 11
depletePoolConstraint .................................................... 13
dummiesToFactor .......................................................... 14
inspectSolution ............................................................ 15
itemCategoryConstraint ................................................... 16
itemCategoryRangeConstraint .......................................... 17
itemExclusionConstraint ................................................ 19
itemsPerFormConstraint .................................................. 21
items_diao ................................................................. 22
items_lsa ................................................................. 22
items_mini .............................................................. 23
items_pilot .............................................................. 24
items_vera .............................................................. 24
itemTuples ............................................................... 25
itemUsageConstraint ..................................................... 26
itemValuesConstraint ..................................................... 27
itemValuesRangeConstraint ............................................. 28
matrixExclusionTuples .................................................. 30
maximinObjective ........................................................ 31
maxObjective ............................................................. 32
minimaxObjective ........................................................ 33
minObjective ............................................................. 34
stemInclusionTuples ....................................................... 35
useSolver ................................................................. 36

Index 38

---

analyzeBlockExclusion  Analyze block exclusiveness

Description

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

Usage

analyzeBlockExclusion(
  solverOut,
  items,
  idCol,
  exclusionTuples,
  formName = "form"
)

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.
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.
formName A character vector with names to give to the forms.

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 <- itemTuples(items = items, idCol = "ID", infoCol = "exclusions",
  sepPattern = ", ")

# Create constraints
exclusion_constraint <- itemExclusionConstraint(nForms = 2, itemTuples = exTuples2, itemIDs = items$ID)
depletion_constraint <- depletePoolConstraint(2, nItems = 4, itemIDs = items$ID)
target_constraint <- minimaxObjective(nForms = 2,
analyzeComplexBlockExclusion

```r
itemValues = c(3, 1.5, 2, 4),
            targetValue = 1,
            itemIDs = items$ID)

  opt_solution <- useSolver(list(exclusion_constraint, target_constraint,
                                  depletion_constraint))

  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**

```r
analyzeComplexBlockExclusion(
  solverOut_list,  # List of objects created by useSolver.
  items_list,      # List of 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.
  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.
)
```

**Arguments**

- `solverOut_list`: List of objects created by `useSolver`.
- `items_list`: List of 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.
- `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

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

### Arguments

- **solverOut**: Object created by useSolver function.
- **items**: Original data.frame containing information on item level.
- **idCol**: Column name or column number 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

```r
## 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, operator = "=",
  targetValue = 1, itemIDs = items$ID)
perForm <- itemsPerFormConstraint(nForms = 2, operator = "=",
  targetValue = 5, itemIDs = items$ID)
```
autoItemValuesMinMaxConstraint

Create single value constraints with minimum and maximum.

Description

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

Usage

autoItemValuesMinMaxConstraint(
  nForms,
  itemValues,
  testLength = NULL,
  allowedDeviation = NULL,
  relative = FALSE,
  verbose = TRUE,
  itemIDs = NULL
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nForms</td>
<td>Number of forms to be created.</td>
</tr>
<tr>
<td>itemValues</td>
<td>Item parameter/values for which the sum per test form should be constrained.</td>
</tr>
<tr>
<td>testLength</td>
<td>To be documented.</td>
</tr>
<tr>
<td>allowedDeviation</td>
<td>Numeric value of length 1. How much deviance is allowed from target values?</td>
</tr>
<tr>
<td>relative</td>
<td>Is the allowedDeviation expressed as a proportion?</td>
</tr>
<tr>
<td>verbose</td>
<td>Should calculated values be reported?</td>
</tr>
<tr>
<td>itemIDs</td>
<td>A character vector of item IDs in correct ordering, or NULL.</td>
</tr>
</tbody>
</table>
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. For detailed documentation on how the minimum and maximum are calculated see also computeTargetValues.

Value

A sparse matrix.

Examples

```r
autoItemValuesMinMaxConstraint(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

```r
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


Examples

```r
# 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**

```r
calculateIIF(A = rep(1, length(B)), B, C = rep(0, length(B)), theta, D = 1.7)
```

**Arguments**

- `A` Vector of discrimination parameters.
- `B` Vector of difficulty parameters.
- `C` Vector of pseudo-guessing parameters.
- `theta` Vector of time intensity parameters.
- `D` the constant that should be used. Defaults to 1.7.

**Value**

A matrix, with columns for different theta and rows for different items.
cappedMaximinObjective

References


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 theta-values (3PL model)
calculateIIF(B = -0.5, C = 0.25, theta = c(-1, 0, 1))

# TIF for multiple items and multiple ability levels (2PL model)
calculateIIF(A = c(0.7, 1.1, 0.8), B = c(1.1, 0.8, 0.5),
   theta = c(-1, 0, 1))

cappedMaximinObjective

CappedMaximin Constraint.

Description

Create maximin-constraints related to an item parameter/value. That is, the created constraints can be used to maximize the minimal sum of the item values (itemValues), while at the same time automatically setting an ideal upper limit to the overflow. More specifically, the capped minimax method described by Luo (2020) is used.

Usage

cappedMaximinObjective(
   nForms,
   itemValues,
   weight = 1,
   whichForms = seq_len(nForms),
   info_text = NULL,
   itemIDs = names(itemValues)
)

Arguments

nForms Number of forms to be created.
itemValues Item parameter/values for which the sum per test form should be constrained.
weight a weight for the real-valued variable(s). Useful when multiple constraints are combined. Should only be used if the implications are well understood.
combineConstraints

whichForms An integer vector indicating which test forms should be constrained. Defaults to all the test forms.

info_text a character string of length 1, to be used in the "info"-attribute of the resulting constraint-object.

itemIDs a character vector of item IDs in correct ordering, or NULL.

Value An object of class "constraint".

References


Examples

# constraint that minimizes the maximum difference per test form value and a # target value of 0
cappedMaximinObjective(nForms = 2, itemValues = rep(-2:2, 2))

combineConstraints Combine constraints

Description

Combine multiple constraint-objects into one constraint object.

Usage

combineConstraints(...) message = TRUE)

Arguments

... multiple constraint-objects or a list with multiple constraint-objects

message A logical indicating whether a message should be given when only one constraint object is combined.

Value

A data.frame of block exclusions.
Examples

```r
combineConstraints(
  itemValuesConstraint(2, 1:10, operator = ">=", targetValue = 4),
  itemValuesConstraint(2, 1:10, operator = "<=", targetValue = 6)
)
```

**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**

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

## Default S3 method:
```r
computeTargetValues(
  itemValues,
  nForms,
  testLength = NULL,
  allowedDeviation = NULL,
  relative = FALSE
)
```

## S3 method for class 'factor'
```r
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.
computeTargetValues

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 information values for an allowed deviation of
## 10 percent are:
computeTargetValues(itemInformations, nForms = 3, allowedDeviation = .10,
                   relative = TRUE, testLength = 10)

## items_vera$MC is dummy variable indication which items in the pool are multiple choice
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

deplicatePoolConstraint(nForms, nItems = NULL, itemIDs = NULL)

Arguments

nForms Number of forms to be created.
nItems Number of items in the item pool [optional to create itemIDs automatically].
itemIDs a character vector of item IDs in correct ordering, or NULL.

Value

A sparse matrix.
**dummiesToFactor**

**Examples**

depletePoolConstraint(2, itemIDs = 1:10)

dummiesToFactor

**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")
inspectSolution

**Description**

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

**Usage**

```r
inspectSolution(
  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 = names(items),  # Which columns should be used from the `items data.frame`?
  colSums = TRUE  # Should column sums be calculated in the output? Only works if all columns are numeric.
)
```

**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 `colSums` 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**

```r
## 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, operator = "=",
```
targetValue = 1, itemIDs = items$ID)
perForm <- itemsPerFormConstraint(nForms = 2, operator = "=",
                      targetValue = 5, itemIDs = items$ID)
target <- minimaxObjective(nForms = 2,
                      itemValues = items$itemValues,
                      targetValue = 0, itemIDs = items$ID)
sol <- useSolver(allConstraints = list(usage, perForm, target),
                      solver = "lpSolve")

## Inspect Solution
out <- inspectSolution(sol, items = items, idCol = 1, 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, itemCategories, operator = c("="), targetValues,
  whichForms = seq_len(nForms), info_text = NULL,
  itemIDs = names(itemCategories)
)

Arguments

nForms Number of forms to be created.
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.
whichForms An integer vector indicating which test forms should be constrained. Defaults to all the test forms.
itemCategoryRangeConstraint

info_text      a character string of length 1, to be used in the "info"-attribute of the resulting constraint-object.

itemIDs       a character vector of item IDs in correct ordering, or NULL.

Value

A object of class "constraint".

Examples

```r
## 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, item_type, "\geq", targetValues = c(1, 3, 2))
```

---

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

```r
itemCategoryRangeConstraint(
  nForms,
  itemCategories,
  range,
  whichForms = seq_len(nForms),
  info_text = NULL,
  itemIDs = names(itemCategories)
)
```

```r
itemCategoryMinConstraint(
  nForms,
  itemCategories,
  min,
  whichForms = seq_len(nForms),
  info_text = NULL,
  itemIDs = names(itemCategories)
)
```
itemCategoryMaxConstraint(
  nForms,
  itemCategories,
  max,
  whichForms = seq_len(nForms),
  info_text = NULL,
  itemIDs = names(itemCategories)
)

itemCategoryDeviationConstraint(
  nForms,
  itemCategories,
  targetValues,
  allowedDeviation,
  relative = FALSE,
  whichForms = seq_len(nForms),
  info_text = NULL,
  itemIDs = names(itemCategories)
)

Arguments

- **nForms**: Number of forms to be created.
- **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
- **whichForms**: An integer vector indicating which test forms should be constrained. Defaults to all the test forms.
- **info_text**: a character string of length 1, to be used in the "info"-attribute of the resulting constraint-object.
- **itemIDs**: a character vector of item IDs in correct ordering, or NULL.
- **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.
itemExclusionConstraint

Value
A sparse matrix.

Functions
• itemCategoryMinConstraint: constrain minimum value
• itemCategoryMaxConstraint: constrain maximum value
• itemCategoryDeviationConstraint: constrain the distance from the targetValues

Examples

```r
## 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))
itemCategoryRangeConstraint(2, item_type, range = cbind(min = rep(2, 3), max = rep(4, 3)))

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

itemExclusionConstraint

Create item inclusion or exclusion constraints.

Description
Create constraints that prohibit that item pairs occur in the same test forms (exclusions) or force item pairs to be in the same test forms (inclusions).

Usage

```r
itemExclusionConstraint(
  nForms, itemTuples, itemIDs,
  whichForms = seq_len(nForms), info_text = NULL
)
```

```r
itemInclusionConstraint(
  nForms, itemTuples, itemIDs,
  whichForms = seq_len(nForms), info_text = NULL
)
```
Arguments

nForms Number of forms to be created.
itemTuples data.frame with two columns, containing tuples with item IDs which should be in test forms inclusively or exclusively.
itemIDs Character vector of item IDs in correct ordering.
whichForms An integer vector indicating which test forms should be constrained. Defaults to all the test forms.
info_text a character string of length 1, to be used in the "info"-attribute of the resulting constraint-object.

Details

Item tuples can, for example, be created by the function itemTuples.

Value

An object of class "constraint".

Functions

- itemExclusionConstraint: item pair exclusion constraints
- itemInclusionConstraint: item pair inclusion constraints

Examples

```r
## Simple Exclusion Example
# item-IDs
IDs <- c("item1", "item2", "item3", "item4")

# exclusion 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)

# inclusion tuples: Items 2 and 3 have to be in the same test form
inTuples <- data.frame(v1 = c("item2"), v2 = c("item3"),
                       stringsAsFactors = FALSE)

# create constraints
itemExclusionConstraint(nForms = 2, itemTuples = exTuples, itemIDs = IDs)
itemInclusionConstraint(nForms = 2, itemTuples = inTuples, itemIDs = IDs)
```

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

# Create tuples
exTuples2 <- itemTuples(items = items, idCol = "ID", infoCol = "infoCol",
                       stringsAsFactors = FALSE)
```

## Create constraints
itemExclusionConstraint(nForms = 2, itemTuples = exTuples2, itemIDs = IDs)

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

### Usage
```r
itemsPerFormConstraint(
  nForms,
  nItems = NULL,
  operator = c("<="", ", ", ", >="),
  targetValue,
  whichForms = seq_len(nForms),
  itemIDs = NULL
)
```

### Arguments
- `nForms`: Number of forms to be created.
- `nItems`: Number of items in the item pool [optional to create itemIDs automatically].
- `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.
- `whichForms`: An integer vector indicating which test forms should be constrained. Defaults to all the test forms.
- `itemIDs`: a character vector of item IDs in correct ordering, or NULL.

### 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
An object of class "constraint".
Examples

```r
## Constrain the test forms to have exactly five items
itemsPerFormConstraint(3, operator = "=" , targetValue = 5,
itemIDs = 1:20)
```

---

**items_diao**

*Small simulated item pool example.*

**Description**

A data.frame containing 165 items calibrated using a 3PL model. This item pool is analogous to one of the item pools used in Diao & van der Linden (2011).

**Usage**

```r
items_diao
```

**Format**

A data.frame.

- **item** Item identifier.
- **a** Discrimination parameter.
- **b** Difficulty parameter.
- **c** Pseudo-guessing parameter.
- **category** Content category.

**References**


---

**items_lsa**

*Simulated item pool example.*

**Description**

A data.frame containing 209 calibrated items with different categorical and metric properties, comparable to an item pool from a large-scale assessment.

**Usage**

```r
items_lsa
```
**Format**

A `data.frame`.

- **testlet**  Testlet identifier (items in the same testlet share a common stimulus).
- **item**  Item identifier.
- **level**  Competence level.
- **format**  Item format.
- **frequency**  Solution frequency.
- **infit**  Item infit.
- **time**  Average response time in seconds.
- **anchor**  Is the item an anchor item?

---

| items_mini | Small simulated item pool example. |

---

**Description**

A `data.frame` containing 30 items with different categorical and metric properties.

**Usage**

`items_mini`

**Format**

A `data.frame`.

- **item**  Item identifier.
- **format**  Item format (e.g., multiple choice, open answer, order item).
- **time**  Average response time in seconds.
- **difficulty**  IRT difficulty parameter.
*items_pilot*  
Small simulated item pool example.

**Description**

A *data.frame* containing 100 not yet calibrated items with different categorical and metric properties.

**Usage**

`items_pilot`

**Format**

A *data.frame*.

- **item** Item identifier.
- **diffCategory** Item difficulty (five categories).
- **format** Item format (multiple choice, constructed multiple choice, or open answer).
- **domain** Item domain (listening, reading, or writing).
- **time** Average response times in seconds.
- **exclusions** Items which cannot be in the same test form.

*items_vera*  
Small artificial item pool example.

**Description**

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

**Usage**

`items_vera`

**Format**

A *data.frame*.

- **item** Item identifier.
- **exclusions** Items which cannot be in the same test form.
- **time** 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, diff_5** Difficulty categories.
Create item tuples.

Description

If item inclusions or exclusions are stored as a character vector, `itemTuples` separates this vector and creates item pairs (‘tuples’).

Usage

```r
itemTuples(items, idCol = "ID", infoCol, sepPattern = ", ")
```

Arguments

- `items` A `data.frame` with information on an item pool.
- `idCol` character or integer indicating the item ID column in `items`.
- `infoCol` character or integer indicating the column in `items` which contains information on the tuples.
- `sepPattern` String which should be used for separating item IDs in the `infoCol` column.

Details

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

Value

A `data.frame` with two columns.

Examples

```r
# Example data.frame
items <- data.frame(ID = c("item1", "item2", "item3", "item4"),
                     exclusions = c("item2, item3", NA, NA, NA))

# Create tuples
itemTuples(items = items, idCol = "ID", infoCol = 2,
            sepPattern = ", ")
```
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 = "\leq"), (b) equal to (operator = "\text{equal}"), or (c) greater or equal than (operator = "\geq") the chosen value.

Usage

```r
itemUsageConstraint(
  nForms,
  nItems = NULL,
  formValues = rep(1, nForms),
  operator = c("\leq", "\text{equal}", "\geq"),
  targetValue = 1,
  whichItems = seq_len(nItems),
  info_text = NULL,
  itemIDs = NULL
)
```

Arguments

- **nForms**: Number of forms to be created.
- **nItems**: Number of items in the item pool [optional to create itemIDs automatically].
- **formValues**: vector with values or weights for each form. Defaults to 1 for each form.
- **operator**: A character indicating which operator should be used in the constraints, with three possible values: "\leq", "\text{equal}", or "\geq". See details for more information.
- **targetValue**: The value to be used in the constraints
- **whichItems**: A vector indicating which items should be constrained. Defaults to all the items.
- **info_text**: a character string of length 1, to be used in the "info"-attribute of the resulting constraint-object.
- **itemIDs**: a character vector of item IDs in correct ordering, or NULL.

Details

When operator = "\leq" 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 = "\text{equal}" and value = 1, each item is used exactly once, which corresponds to no item-overlap and complete item pool depletion.

If certain items are required in the resulting test form(s), as for example anchor items, whichItems can be used to constrain the usage of these items to be exactly 1. whichItems can either be a numeric vector with item numbers or a character vector with item identifiers corresponding to itemIDs.
Value

An object of class "constraint".

Examples

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

## force certain items to be in the test, others not
usage1 <- itemUsageConstraint(2, operator = "=" , targetValue = 1,
itemIDs = paste0("item", 1:20))
usage2 <- itemUsageConstraint(2, operator = "=" , targetValue = 1,
itemIDs = paste0("item", 1:20),
whichItems = c("item5", "item8", "item10"))
```

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

```r
itemValuesConstraint(  
nForms,  
itemValues,  
operator = c("<=" , "=" ,">="),  
targetValue,  
whichForms = seq_len(nForms),  
info_text = NULL,  
itemIDs = names(itemValues)
)
```

Arguments

- `nForms` Number of forms to be created.
- `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: "<=" , "=" ,">=". See details for more information.
- `targetValue` the target test form value.
whichForms  An integer vector indicating which test forms should be constrained. Defaults to all the test forms.
info_text  a character string of length 1, to be used in the "info"-attribute of the resulting constraint-object.
itemIDs  a character vector of item IDs in correct ordering, or NULL.

Value
An object of class "constraint".

Examples
## constraints to make sure that the sum of the item values (1:10) is between
## 4 and 6
combineConstraints(
  itemValuesConstraint(2, 1:10, operator = "\geq", targetValue = 4),
  itemValuesConstraint(2, 1:10, operator = "\leq", targetValue = 6)
)

itemValuesRangeConstraint
Create single value constraints with minimum and maximum.

Description
itemValuesRangeConstraint, itemValuesMinConstraint, and itemValuesMaxConstraint 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
itemValuesRangeConstraint(
  nForms,  
  itemValues,  
  range,  
  whichForms = seq_len(nForms),  
  info_text = NULL,  
  itemIDs = names(itemValues)
)

itemValuesMinConstraint(
  nForms,  
  itemValues,  
  min,  
  whichForms = seq_len(nForms),  
  info_text = NULL,
itemValuesRangeConstraint

    itemIDs = names(itemValues)
)

itemValuesMaxConstraint(
    nForms,  
    itemValues,  
    max,  
    whichForms = seq_len(nForms),  
    info_text = NULL,  
    itemIDs = names(itemValues)
)

itemValuesDeviationConstraint(
    nForms,  
    itemValues,  
    targetValue,  
    allowedDeviation,  
    relative = FALSE,  
    whichForms = seq_len(nForms),  
    info_text = NULL,  
    itemIDs = names(itemValues)
)

Arguments

nForms Number of forms to be created.
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
whichForms An integer vector indicating which test forms should be constrained. Defaults to all the test forms.
info_text a character string of length 1, to be used in the "info"-attribute of the resulting constraint-object.
itemIDs a character vector of item IDs in correct ordering, or NULL.
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

itemValuesDeviationConstraint 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.
matrixExclusionTuples

**Value**

An object of class "constraint".

**Functions**

- `itemValuesMinConstraint`: constrain minimum value
- `itemValuesMaxConstraint`: constrain maximum value
- `itemValuesDeviationConstraint`: constrain the distance from the `targetValue`

**Examples**

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

## or alternatively
itemValuesDeviationConstraint(2, 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.
maximinObjective

Examples

```r
# 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)
```

maximinObjective  Maximin Constraint.

Description

Create maximin-constraints related to an item parameter/value. That is, the created constraints can be used to maximize the minimal sum of the item values (itemValues), while at the same time setting an upper limit to the overflow by means of a maximally allowed deviation allowedDeviation.

Usage

```r
maximinObjective(
  nForms,
  itemValues,
  allowedDeviation,
  weight = 1,
  whichForms = seq_len(nForms),
  info_text = NULL,
  itemIDs = names(itemValues)
)
```

Arguments

- `nForms`: Number of forms to be created.
- `itemValues`: Item parameter/values for which the sum per test form should be constrained.
- `allowedDeviation`: the maximum allowed deviation between the sum of the target values.
- `weight`: a weight for the real-valued variable(s). Useful when multiple constraints are combined. Should only be used if the implications are well understood.
- `whichForms`: An integer vector indicating which test forms should be constrained. Defaults to all the test forms.
- `info_text`: a character string of length 1, to be used in the "info"-attribute of the resulting constraint-object.
- `itemIDs`: a character vector of item IDs in correct ordering, or NULL.
**maxObjective**

**Value**

An object of class "constraint".

**Examples**

```
# constraint that minimizes the maximum difference per test form value and a
# target value of 0
maximinObjective(nForms = 2, itemValues = rep(-2:2, 2),
    allowedDeviation = 1)
```

---

**maxObjective**  
*Max Constraint.*

**Description**

Create max-constraints related to an item parameter/value. That is, the created constraints can be used to maximize the sum of the item values (itemValues) of the test form. Note that this constraint can only be used when only one test form has to be assembled.

**Usage**

```
maxObjective(
    nForms,
    itemValues,
    weight = 1,
    whichForms = seq_len(nForms),
    info_text = NULL,
    itemIDs = names(itemValues)
)
```

**Arguments**

- **nForms**: Number of forms to be created.
- **itemValues**: Item parameter/values for which the sum per test form should be constrained.
- **weight**: a weight for the real-valued variable(s). Useful when multiple constraints are combined. Should only be used if the implications are well understood.
- **whichForms**: An integer vector indicating which test forms should be constrained. Defaults to all the test forms.
- **info_text**: a character string of length 1, to be used in the "info"-attribute of the resulting constraint-object.
- **itemIDs**: a character vector of item IDs in correct ordering, or NULL.

**Value**

An object of class "constraint".
minimaxObjective

Examples

    # constraint that maximizes the sum of the itemValues
    maxObjective(nForms = 1, itemValues = rep(-2:2, 2))

minimaxObjective  Minimax Constraint.

Description

Create minimax-constraints related to an item parameter/value. That is, the created constraints can be used to minimize the maximum distance between the sum of the item values (itemValues) per test form and the chosen targetValue.

Usage

minimaxObjective(
    nForms, itemValues, targetValue, weight = 1, whichForms = seq_len(nForms), info_text = NULL, itemIDs = names(itemValues)
)

Arguments

    nForms  Number of forms to be created.
    itemValues  Item parameter/values for which the sum per test form should be constrained.
    targetValue  the target test form value.
    weight  a weight for the real-valued variable(s). Useful when multiple constraints are combined. Should only be used if the implications are well understood.
    whichForms  An integer vector indicating which test forms should be constrained. Defaults to all the test forms.
    info_text  a character string of length 1, to be used in the "info"-attribute of the resulting constraint-object.
    itemIDs  a character vector of item IDs in correct ordering, or NULL.

Value

An object of class "constraint".
Examples

```r
# constraint that minimizes the maximum difference per test form value and a
target value of 0
minimaxObjective(nForms = 2,
    itemValues = rep(-2:2, 2),
    targetValue = 0)
```

Description

Create min-constraints related to an item parameter/value. That is, the created constraints can be used to minimize the sum of the item values (itemValues) of the test form. Note that this constraint can only be used when only one test form has to be assembled.

Usage

```r
minObjective(
    nForms,
    itemValues,
    weight = 1,
    whichForms = seq_len(nForms),
    info_text = NULL,
    itemIDs = names(itemValues)
)
```

Arguments

- **nForms**: Number of forms to be created.
- **itemValues**: Item parameter/values for which the sum per test form should be constrained.
- **weight**: a weight for the real-valued variable(s). Useful when multiple constraints are combined. Should only be used if the implications are well understood.
- **whichForms**: An integer vector indicating which test forms should be constrained. Defaults to all the test forms.
- **info_text**: a character string of length 1, to be used in the "info"-attribute of the resulting constraint-object.
- **itemIDs**: a character vector of item IDs in correct ordering, or NULL.

Value

An object of class "constraint".
stemInclusionTuples

Examples

# constraint that maximizes the sum of the itemValues
maxObjective(nForms = 1, itemValues = rep(-2:2, 2))

Description

If item-stimulus hierarchies are stored in a single stimulus column, stemInclusionTuples transforms this format into item pairs ('tuples').

Usage

stemInclusionTuples(items, idCol = "ID", stemCol)

Arguments

items A data.frame with information on an item pool.
idCol character or integer indicating the item ID column in items.
stemCol A column in items containing the item stems or stimulus names, shared among items which should be in the same test form.

Details

Inclusion tuples can be used by itemInclusionConstraint to set up inclusion constraints.

Value

A data.frame with two columns.

Examples

# Example data.frame
inclDF <- data.frame(ID = paste0("item_{1:6}, stem = c(rep("stim_1", 3), "stim_3", "stim_4", "stim_3"),
                        stringsAsFactors = FALSE)

# Create tuples
stemInclusionTuples(inclDF, idCol = "ID", stemCol = "stem")
useSolver  

*Use a solver for a list of constraints.*

**Description**

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

**Usage**

```r
useSolver(
  allConstraints,
  solver = c("GLPK", "lpSolve", "Gurobi", "Symphony"),
  timeLimit = Inf,
  formNames = NULL,
  ...
)
```

**Arguments**

- `allConstraints` List of constraints.
- `solver` A character string indicating the solver to use.
- `timeLimit` The maximal runtime in seconds.
- `formNames` A character vector with names to give to the forms.
- `...` 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_found` Was a solution found?
- `solution` Numeric vector containing the found solution.
- `solution_status` Was the solution optimal?
Examples

nForms <- 2
nItems <- 4

# create constraints
target <- minimaxObjective(nForms = nForms, c(1, 0.5, 1.5, 2),
                        targetValue = 2, itemIDs = 1:nItems)
noItemOverlap <- itemUsageConstraint(nForms, operator = "=" , itemIDs = 1:nItems)
testLength <- itemsPerFormConstraint(nForms = nForms,
                        operator = "<=", targetValue = 2, itemIDs = 1:nItems)

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

* datasets
  items_diao, 22
  items_lsa, 22
  items_mini, 23
  items_pilot, 24
  items_vera, 24

analyzeBlockExclusion, 2
analyzeComplexBlockExclusion, 4
appendSolution, 5
autoItemValuesMinMaxConstraint, 6

calculateExpectedRT, 7
calculateIIF, 8
cappedMaximinObjective, 9
combineConstraints, 10
computeTargetValues, 7, 11
depletePoolConstraint, 13
dummiesToFactor, 14

ingressSolution, 15
itemCategoryConstraint, 16
itemCategoryDeviationConstraint
  (itemCategoryRangeConstraint), 17
itemCategoryMaxConstraint
  (itemCategoryRangeConstraint), 17
itemCategoryMinConstraint
  (itemCategoryRangeConstraint), 17
itemCategoryRangeConstraint, 17
itemExclusionConstraint, 3, 4, 19, 25, 30
itemInclusionConstraint, 25, 35
itemInclusionConstraint
  (itemExclusionConstraint), 19
items_diao, 22
items_lsa, 22
items_mini, 23
items_pilot, 24
items_vera, 24
itemsPerFormConstraint, 21
itemTuples, 20, 25
itemUsageConstraint, 26
itemValuesConstraint, 27
itemValuesDeviationConstraint, 6
itemValuesDeviationConstraint
  (itemValuesRangeConstraint), 28
itemValuesMaxConstraint
  (itemValuesRangeConstraint), 28
itemValuesMinConstraint
  (itemValuesRangeConstraint), 28
itemValuesRangeConstraint, 28
lp, 36

dmatrixExclusionTuples, 30
maximinObjective, 31
maxObjective, 32
minimaxObjective, 33
minObjective, 34
Rglpk_solve_LP, 36
stemInclusionTuples, 35
useSolver, 36