Package: TestDesign (via r-universe)

October 8, 2026

Type Package

Title Optimal Test Design Approach to Fixed and Adaptive Test Construction

Version 1.7.1

Maintainer Seung W. Choi <schoi@austin.utexas.edu>

Description Uses the optimal test design approach by Birnbaum (1968, ISBN:9781593119348) and van der Linden (2018) <doi:10.1201/9781315117430> to construct fixed, adaptive, and parallel tests. Supports the following mixed-integer programming (MIP) solver packages: 'Rsymphony', 'highs', 'gurobi', 'lpSolve', and 'Rglpk'. The 'gurobi' package is not available from CRAN; see <https://www.gurobi.com/downloads/>.

URL https://choi-phd.github.io/TestDesign/ (documentation)

BugReports https://github.com/choi-phd/TestDesign/issues/

License GPL (>= 2)

Depends R (>= 4.0)

Imports Rcpp (>= 1.0.0), methods, lpSolve, foreach, logitnorm, crayon, highs

SystemRequirements C++17

Suggests Rsymphony, gurobi, Rglpk, mirt, mirtCAT, progress, shiny, shinythemes, shinyWidgets, shinyjs, DT, knitr, rmarkdown, kableExtra, testthat (>= 2.1.0), pkgdown, pkgload

LinkingTo Rcpp, RcppArmadillo

Encoding UTF-8

LazyData true

VignetteBuilder knitr

Collate 'RcppExports.R' 'import.R' 'extensions.R' 'item_class.R' 'calc_prob_functions.r' 'calc_escore_functions.r' 'calc_location_functions.r' 'calc_fisher_functions.r' 'calc_loglikelihood_functions.r' 'calc_jacobian_functions.r'

'calc_hessian_functions.r' 'sim_resp_functions.r'
'loading_functions.R' 'static_class.R' 'shadow_class.R'
'item_pool_operators.R' 'item_attrib_operators.R'
'st_attrib_operators.R' 'constraints_operators.R'
'static_functions.R' 'shadow_functions.R' 'bayes_functions.R'
'calculate_adaptivity_measures.r' 'constraint_functions.R'
'cpp_calc_documents.r' 'cpp_core_documents.r'
'cpp_theta_documents.r' 'datasets.R' 'eligibility_functions.R'
'exposure_control_functions.R' 'solver_functions.R'
'helper_functions.R' 'item_pool_cluster_operators.R'
'other_functions.R' 'partitioning_class.r'
'partitioning_functions.r' 'plot_functions.R' 'summary_class.R'
'print_functions.R' 'runshiny.R' 'shadowtest_functions.R'
'summary_functions.R' 'show_functions.R'
'simulation_data_cache_class.r'
'simulation_data_cache_operators.r' 'theta_functions.R'
'xdata_functions.R'

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Seung W. Choi [aut, cre] (ORCID:
<https://orcid.org/0000-0003-4777-5420>), Sangdon Lim [aut]
(ORCID: <https://orcid.org/0000-0002-2988-014X>)

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

Date/Publication 2026-09-08 09:50:30 UTC

RemoteUrl https://github.com/cran/TestDesign

RemoteRef HEAD

RemoteSha d4505627291a65cd1c4a8bd660329e7f4f5f3549

Contents

a_to_alpha 4
app 5
buildConstraints 5
calc_info 6
calc_info_EB 8
calc_info_FB 9
calc_likelihood 9
calc_MI_FB 11
calc_posterior 12
calc_posterior_function 12
calc_posterior_single 13
calcEscore 14
calcFisher 16
calcHessian 18
calcJacobian 21

calcLocation-methods . . . 23
calcLogLikelihood . . . 25
calcProb-methods . . . 26
calculateAdaptivityMeasures . . . 29
checkConstraints . . . 30
config_Shadow-class . . . 30
config_Static-class . . . 34
constraint-class . . . 36
constraints-class . . . 37
constraints-operators . . . 38
dataset_bayes . . . 39
dataset_fatigue . . . 39
dataset_reading . . . 40
dataset_science . . . 41
detectBestSolver . . . 41
e_item . . . 42
eap . . . 44
find_segment . . . 46
getScoreAttributes . . . 46
getSolution . . . 47
getSolutionAttributes . . . 48
h_item . . . 49
info_item . . . 51
iparPosteriorSample . . . 55
item-classes . . . 56
item_attrib-class . . . 57
item_attrib-operators . . . 58
item_pool-class . . . 59
item_pool-operators . . . 59
item_pool_cluster-class . . . 61
j_item . . . 61
lnHyperPars . . . 63
loadConstraints . . . 64
loadItemPool . . . 65
logitHyperPars . . . 66
makeConstraintsByEachPartition . . . 67
makeItemPoolCluster . . . 68
makeSimulationDataCache . . . 68
makeTest . . . 69
makeTestCluster . . . 70
mle . . . 71
mlef . . . 73
output_Shadow-class . . . 75
output_Shadow_all-class . . . 76
output_Split-class . . . 77
output_Static-class . . . 78
p_item . . . 78
plot . . . 81

print . . . 86
RE . . . 87
RMSE . . . 88
Shadow . . . 88
show . . . 90
simResp . . . 92
simulation_data_cache-class . . . 94
Split . . . 95
st_attrib-class . . . 96
st_attrib-operators . . . 97
Static . . . 98
summary . . . 99
summary-classes . . . 100
test-class . . . 101
test_cluster-class . . . 101
test_operators . . . 102
TestDesign . . . 102
testSolver . . . 103
theta_EAP . . . 103
theta_EB . . . 105
theta_FB . . . 106
toggleConstraints . . . 108

Index **109**

`a_to_alpha` *Calculate alpha angles from a-parameters*

Description

`a_to_alpha` is a function for converting an a-parameter vector to an alpha angle vector. The returned values are in the radian metric.

Usage

```
a_to_alpha(a)
```

Arguments

`a`	the *a*-parameter vector.

Examples

```
a_to_alpha(c(1, 1))
```

app *Open TestDesign app*

Description

app and OAT are aliases of TestDesign.

Usage

```
app()

OAT()
```

Details

TestDesign is a caller function for opening the Shiny interface of TestDesign package.

Examples

```
## Not run:
if (interactive()) {
  TestDesign()
}

## End(Not run)
```

buildConstraints *Build constraints (shortcut to other loading functions)*

Description

buildConstraints is a data loading function for creating a constraints object. buildConstraints is a shortcut that calls other data loading functions. The constraints must be in the expected format; see the vignette in vignette("constraints").

Usage

```
buildConstraints(object, item_pool, item_attrib, st_attrib = NULL)
```

Arguments

object	constraint specifications. Can be a data.frame or the file path of a .csv file. See the vignette for the expected format.
item_pool	item parameters. Can be a item_pool object, a data.frame or the file path of a .csv file.
item_attrib	item attributes. Can be an item_attrib object, a data.frame or the file path of a .csv file.
st_attrib	(optional) stimulus attributes. Can be an st_attrib object, a data.frame or the file path of a .csv file.

Value

buildConstraints returns a constraints object. This object is used in Static and Shadow.

Examples

```
## Read from objects:
constraints_science <- buildConstraints(constraints_science_data,
  itempool_science, itemattrib_science)
constraints_reading <- buildConstraints(constraints_reading_data,
  itempool_reading, itemattrib_reading, stimattrib_reading)

## Read from data.frame:
constraints_science <- buildConstraints(constraints_science_data,
  itempool_science_data, itemattrib_science_data)
constraints_reading <- buildConstraints(constraints_reading_data,
  itempool_reading_data, itemattrib_reading_data, stimattrib_reading_data)

## Read from file: write to tempdir() for illustration and clean afterwards
f1 <- file.path(tempdir(), "constraints_science.csv")
f2 <- file.path(tempdir(), "itempool_science.csv")
f3 <- file.path(tempdir(), "itemattrib_science.csv")
write.csv(constraints_science_data, f1, row.names = FALSE)
write.csv(itempool_science_data   , f2, row.names = FALSE)
write.csv(itemattrib_science_data , f3, row.names = FALSE)
constraints_science <- buildConstraints(f1, f2, f3)
file.remove(f1)
file.remove(f2)
file.remove(f3)
```

---

calc_info *(C++) For multiple items, calculate Fisher information*

Description

calc_info() and calc_info_matrix() are functions for calculating Fisher information. These functions are designed for multiple items.

Usage

```
calc_info(x, item_parm, ncat, model)

calc_info_matrix(x, item_parm, ncat, model)
```

Arguments

`x`	the theta value. This must be a column vector in matrix form for `calc_info_matrix()`.
`item_parm`	a matrix containing item parameters. Each row should represent an item.
`ncat`	a vector containing the number of response categories of each item.
`model`	a vector indicating item models of each item, using • `1`: 1PL model • `2`: 2PL model • `3`: 3PL model • `4`: PC model • `5`: GPC model • `6`: GR model

Details

`calc_info()` accepts a single theta value, and `calc_info_matrix()` accepts multiple theta values.

Currently supports unidimensional models.

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
# item parameters
item_parm <- matrix(c(
  1, NA,   NA,
  1,  2,   NA,
  1,  2, 0.25,
  0,  1,   NA,
  2,  0,    1,
  2,  0,    2),
  nrow = 6,
  byrow = TRUE
)

ncat  <- c(2, 2, 2, 3, 3, 3)
model <- c(1, 2, 3, 4, 5, 6)

# single theta example
x <- 0.5
calc_info(x, item_parm, ncat, model)

# multiple thetas example
x <- matrix(seq(0.1, 0.5, 0.1)) # column vector in matrix form
calc_info_matrix(x, item_parm, ncat, model)
```

`calc_info_EB` *Calculate the Fisher information using empirical Bayes*

Description

Calculate the Fisher information using empirical Bayes.

Usage

```
calc_info_EB(x, item_parm, ncat, model)
```

Arguments

`x`	A numeric vector of MCMC sampled theta values.
`item_parm`	A numeric matrix of item parameters.
`ncat`	a numeric vector specifying the number of response categories in each item.
`model`	a numeric vector indicating the IRT models of each item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).

calc_info_FB *Calculate the Fisher information using full Bayesian*

Description

Calculate the Fisher information using full Bayesian.

Usage

```
calc_info_FB(x, items_list, ncat, model, useEAP = FALSE)
```

Arguments

x	A numeric vector of MCMC sampled theta values.
items_list	A list of item parameter matrices.
ncat	a numeric vector specifying the number of response categories in each item.
model	a numeric vector indicating the IRT models of each item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).
useEAP	TRUE to use the mean of MCMC theta draws.

calc_likelihood *(C++) For multiple items, calculate likelihoods*

Description

`calc_likelihood()` and `calc_likelihood_function()` are functions for calculating likelihoods.

Usage

```
calc_likelihood(x, item_parm, resp, ncat, model)

calc_likelihood_function(theta_grid, item_parm, resp, ncat, model)

calc_log_likelihood(x, item_parm, resp, ncat, model, prior, prior_parm)

calc_log_likelihood_function(
  theta_grid,
  item_parm,
  resp,
  ncat,
  model,
  prior,
  prior_parm
)
```

Arguments

x, theta_grid	the theta value. This must be a column vector in matrix form for calc_*_function() functions.
item_parm	a matrix containing item parameters. Each row should represent an item.
resp	a vector containing responses on each item.
ncat	a vector containing the number of response categories of each item.
model	a vector indicating item models of each item, using • 1: 1PL model • 2: 2PL model • 3: 3PL model • 4: PC model • 5: GPC model • 6: GR model
prior	an integer indicating the type of prior distribution, using • 1: normal distribution • 2: uniform distribution
prior_parm	a vector containing parameters for the prior distribution.

Details

calc_log_likelihood() and calc_log_likelihood_function() are functions for calculating log likelihoods.

These functions are designed for multiple items.

calc_*() functions accept a single theta value, and calc_*_function() functions accept multiple theta values.

Currently supports unidimensional models.

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
# item parameters
item_parm <- matrix(c(
  1, NA,   NA,
  1,  2,   NA,
  1,  2, 0.25,
  0,  1,   NA,
  2,  0,    1,
  2,  0,    2),
  nrow = 6,
  byrow = TRUE
)

ncat  <- c(2, 2, 2, 3, 3, 3)
model <- c(1, 2, 3, 4, 5, 6)
resp  <- c(0, 1, 0, 1, 0, 1)

x <- 3
l  <- calc_likelihood(x, item_parm, resp, ncat, model)
ll <- calc_log_likelihood(x, item_parm, resp, ncat, model, 2, NA)
log(l) == ll

x <- matrix(seq(-3, 3, .1))
l  <- calc_likelihood_function(x, item_parm, resp, ncat, model)
ll <- calc_log_likelihood_function(x, item_parm, resp, ncat, model, 2, NA)
all(log(l) == ll)
```

`calc_MI_FB` *Calculate the mutual information using full Bayesian*

Description

Calculate the mutual information using full Bayesian.

Usage

```
calc_MI_FB(x, items_list, ncat, model)
```

Arguments

x	A numeric vector of MCMC sampled theta values.
items_list	A list of item parameter matrices.
ncat	a numeric vector specifying the number of response categories in each item.
model	a numeric vector indicating the IRT models of each item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).

---

calc_posterior *Calculate a posterior value of theta*

Description

Calculate a posterior value of theta.

Usage

```
calc_posterior(x, item_parm, resp, ncat, model, prior, prior_parm)
```

Arguments

x	A length-one numeric vector for a theta value.
item_parm	A numeric matrix of item parameters.
resp	a numeric vector containing item responses.
ncat	A numeric vector of the number of response categories by item.
model	A numeric vector indicating the IRT models of each item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).
prior	The type of prior distribution (1: normal, 2: uniform).
prior_parm	A numeric vector of hyperparameters for the prior distribution, c(mu, sigma) or c(ll, ul).

---

calc_posterior_function *Calculate a posterior distribution of theta*

Description

Calculate a posterior distribution of theta.

Usage

```
calc_posterior_function(
  theta_grid,
  item_parm,
  resp,
  ncat,
  model,
  prior,
  prior_parm
)
```

Arguments

theta_grid	An equi-spaced grid of theta values.
item_parm	A numeric matrix of item parameters.
resp	a numeric vector containing item responses.
ncat	A numeric vector of the number of response categories by item.
model	A numeric vector indicating the IRT models of each item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).
prior	The type of prior distribution (1: normal, 2: uniform).
prior_parm	A numeric vector of hyperparameters for the prior distribution, c(mu, sigma) or c(ll, ul).

---

calc_posterior_single *Calculate a posterior value of theta for a single item*

Description

Calculate a posterior value of theta for a single item.

Usage

```
calc_posterior_single(x, item_parm, resp, ncat, model, prior, prior_parm)
```

Arguments

x	A length-one numeric vector for a theta value.
item_parm	A numeric vector of item parameters (for one item).
resp	A length-one numeric vector of item responses.
ncat	A length-one numeric vector of the number of response categories by item.
model	A length-one numeric vector of the IRT model by item (1: 1PL, 2: 2PL, 3: 3PL, 4: PC, 5: GPC, 6: GR).
prior	The type of prior distribution (1: normal, 2: uniform).
prior_parm	A numeric vector of hyperparameters for the prior distribution, c(mu, sigma) or c(ll, ul).

calcEscore *Calculate expected scores*

Description

calcEscore is a function for calculating expected scores.

Usage

```
calcEscore(object, theta)

## S4 method for signature 'item_1PL,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_2PL,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_3PL,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_PC,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_GPC,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_GR,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_pool,numeric'
calcEscore(object, theta)

## S4 method for signature 'item_1PL,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_2PL,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_3PL,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_PC,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_GPC,matrix'
calcEscore(object, theta)
```

```
## S4 method for signature 'item_GR,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_pool,matrix'
calcEscore(object, theta)

## S4 method for signature 'item_pool_cluster,numeric'
calcEscore(object, theta)
```

Arguments

object	an item or an item_pool object.
theta	theta values to use.

Value

item **object:** calcEscore a vector containing expected score of the item at the theta values.

item_pool **object:** calcEscore returns a vector containing the pool-level expected score at the theta values.

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
item_1     <- new("item_1PL", difficulty = 0.5)
item_2     <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3     <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4     <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5     <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
item_6     <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

ICC_item_1 <- calcEscore(item_1, seq(-3, 3, 1))
ICC_item_2 <- calcEscore(item_2, seq(-3, 3, 1))
ICC_item_3 <- calcEscore(item_3, seq(-3, 3, 1))
ICC_item_4 <- calcEscore(item_4, seq(-3, 3, 1))
ICC_item_5 <- calcEscore(item_5, seq(-3, 3, 1))
ICC_item_6 <- calcEscore(item_6, seq(-3, 3, 1))
TCC_pool   <- calcEscore(itempool_science, seq(-3, 3, 1))
```

`calcFisher` *Calculate Fisher information*

Description

`calcFisher` is a function for calculating Fisher information.

Usage

```
calcFisher(object, theta)

## S4 method for signature 'item_1PL,numeric'
calcFisher(object, theta)

## S4 method for signature 'item_2PL,numeric'
calcFisher(object, theta)

## S4 method for signature 'item_3PL,numeric'
calcFisher(object, theta)

## S4 method for signature 'item_PC,numeric'
calcFisher(object, theta)

## S4 method for signature 'item_GPC,numeric'
calcFisher(object, theta)

## S4 method for signature 'item_GR,numeric'
calcFisher(object, theta)

## S4 method for signature 'item_pool,numeric'
calcFisher(object, theta)
```

```
## S4 method for signature 'item_1PL,matrix'
calcFisher(object, theta)

## S4 method for signature 'item_2PL,matrix'
calcFisher(object, theta)

## S4 method for signature 'item_3PL,matrix'
calcFisher(object, theta)

## S4 method for signature 'item_PC,matrix'
calcFisher(object, theta)

## S4 method for signature 'item_GPC,matrix'
calcFisher(object, theta)

## S4 method for signature 'item_GR,matrix'
calcFisher(object, theta)

## S4 method for signature 'item_pool,matrix'
calcFisher(object, theta)

## S4 method for signature 'item_pool_cluster,numeric'
calcFisher(object, theta)
```

Arguments

object	an item or an item_pool object.
theta	theta values to use.

Value

item **object:** calcFisher returns a (*nq*, *1*) matrix of information values.

item_pool **object:** calcProb returns a (*nq*, *ni*) matrix of information values.

notations • *nq* denotes the number of theta values.
• *ni* denotes the number of items in the item_pool object.

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
item_1      <- new("item_1PL", difficulty = 0.5)
item_2      <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3      <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4      <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5      <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
item_6      <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

info_item_1 <- calcFisher(item_1, seq(-3, 3, 1))
info_item_2 <- calcFisher(item_2, seq(-3, 3, 1))
info_item_3 <- calcFisher(item_3, seq(-3, 3, 1))
info_item_4 <- calcFisher(item_4, seq(-3, 3, 1))
info_item_5 <- calcFisher(item_5, seq(-3, 3, 1))
info_item_6 <- calcFisher(item_6, seq(-3, 3, 1))
info_pool   <- calcFisher(itempool_science, seq(-3, 3, 1))
```

---

calcHessian *Calculate second derivative of log-likelihood*

Description

calcHessian is a function for calculating the second derivative of the log-likelihood function.

Usage

```
calcHessian(object, theta, resp)

## S4 method for signature 'item_1PL,numeric,numeric'
calcHessian(object, theta, resp)
```

```
## S4 method for signature 'item_2PL,numeric,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_3PL,numeric,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_PC,numeric,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_GPC,numeric,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_GR,numeric,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_1PL,matrix,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_2PL,matrix,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_3PL,matrix,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_PC,matrix,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_GPC,matrix,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_GR,matrix,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_pool,numeric,numeric'
calcHessian(object, theta, resp)

## S4 method for signature 'item_pool_cluster,numeric,list'
calcHessian(object, theta, resp)
```

Arguments

object	an item or an item_pool object.
theta	theta values to use.
resp	the response data to use. This must be a single value for an item, or a length *ni* vector for an item_pool.

Details

notations • *nq* denotes the number of theta values.
• *ni* denotes the number of items in the `item_pool` object.

Value

`item` **object:** `calcHessian` returns a length *nq* vector containing the second derivative of the log-likelihood function, of observing the response at each theta.

`item_pool` **object:** `calcHessian` returns a (*nq*, *ni*) matrix containing the second derivative of the log-likelihood function, of observing the response at each theta.

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
item_1    <- new("item_1PL", difficulty = 0.5)
item_2    <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3    <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4    <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5    <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
item_6    <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

h_item_1 <- calcHessian(item_1, seq(-3, 3, 1), 0)
h_item_2 <- calcHessian(item_2, seq(-3, 3, 1), 0)
```

```
h_item_3 <- calcHessian(item_3, seq(-3, 3, 1), 0)
h_item_4 <- calcHessian(item_4, seq(-3, 3, 1), 0)
h_item_5 <- calcHessian(item_5, seq(-3, 3, 1), 0)
h_item_6 <- calcHessian(item_6, seq(-3, 3, 1), 0)
h_pool   <- calcHessian(
  itempool_science, seq(-3, 3, 1),
  rep(0, itempool_science@ni)
)
```

---

calcJacobian *Calculate first derivative of log-likelihood*

Description

calcJacobian is a function for calculating the first derivative of the log-likelihood function.

Usage

```
calcJacobian(object, theta, resp)

## S4 method for signature 'item_1PL,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_2PL,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_3PL,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_PC,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_GPC,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_GR,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_1PL,matrix,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_2PL,matrix,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_3PL,matrix,numeric'
calcJacobian(object, theta, resp)
```

```
## S4 method for signature 'item_PC,matrix,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_GPC,matrix,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_GR,matrix,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_pool,numeric,numeric'
calcJacobian(object, theta, resp)

## S4 method for signature 'item_pool_cluster,numeric,list'
calcJacobian(object, theta, resp)
```

Arguments

`object`	an `item` or an `item_pool` object.
`theta`	theta values to use.
`resp`	the response value to use for each item.

Value

`item` **object:** `calcJacobian` returns a length *nq* vector containing the first derivative of the log-likelihood function, of observing the response at each theta.

`item_pool` **object:** `calcJacobian` returns a (*nq*, *ni*) matrix containing the first derivative of the log-likelihood function, of observing the response at each theta.

notations

- *nq* denotes the number of theta values.
- *ni* denotes the number of items in the `item_pool` object.

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
item_1    <- new("item_1PL", difficulty = 0.5)
item_2    <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3    <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4    <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5    <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
item_6    <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

j_item_1 <- calcJacobian(item_1, seq(-3, 3, 1), 0)
j_item_2 <- calcJacobian(item_2, seq(-3, 3, 1), 0)
j_item_3 <- calcJacobian(item_3, seq(-3, 3, 1), 0)
j_item_4 <- calcJacobian(item_4, seq(-3, 3, 1), 0)
j_item_5 <- calcJacobian(item_5, seq(-3, 3, 1), 0)
j_item_6 <- calcJacobian(item_6, seq(-3, 3, 1), 0)
j_pool   <- calcJacobian(
  itempool_science, seq(-3, 3, 1),
  rep(0, itempool_science@ni)
)
```

---

calcLocation-methods *Calculate central location (overall difficulty)*

Description

calcLocation is a function for calculating the central location (overall difficulty) of items.

Usage

```
calcLocation(object)

## S4 method for signature 'item_1PL'
calcLocation(object)

## S4 method for signature 'item_2PL'
calcLocation(object)

## S4 method for signature 'item_3PL'
calcLocation(object)
```

```
## S4 method for signature 'item_PC'
calcLocation(object)

## S4 method for signature 'item_GPC'
calcLocation(object)

## S4 method for signature 'item_GR'
calcLocation(object)

## S4 method for signature 'item_pool'
calcLocation(object)
```

Arguments

object	an item or an item_pool object.

Value

item **object:** calcLocation returns a theta value representing the central location.

item_pool **object:** calcProb returns a length *ni* list, each containing the central location of the item.

notations • *ni* denotes the number of items in the item_pool object.

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
item_1      <- new("item_1PL", difficulty = 0.5)
item_2      <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3      <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4      <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5      <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
item_6      <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

loc_item_1 <- calcLocation(item_1)
loc_item_2 <- calcLocation(item_2)
loc_item_3 <- calcLocation(item_3)
loc_item_4 <- calcLocation(item_4)
loc_item_5 <- calcLocation(item_5)
loc_item_6 <- calcLocation(item_6)
loc_pool   <- calcLocation(itempool_science)
```

---

calcLogLikelihood *Calculate log-likelihood*

Description

calcLogLikelihood is a function for calculating log-likelihood values.

Usage

```
calcLogLikelihood(object, theta, resp)

## S4 method for signature 'item_pool,numeric,numeric'
calcLogLikelihood(object, theta, resp)

## S4 method for signature 'item_pool,numeric,matrix'
calcLogLikelihood(object, theta, resp)

## S4 method for signature 'item_pool,matrix,numeric'
calcLogLikelihood(object, theta, resp)

## S4 method for signature 'item_pool,matrix,matrix'
calcLogLikelihood(object, theta, resp)
```

Arguments

object	an item_pool object.
theta	theta values to use.
resp	the response data to use.

Value

calcLogLikelihood returns values of log-likelihoods.

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
j_pool   <- calcLogLikelihood(itempool_science, seq(-3, 3, 1), 0)
```

---

calcProb-methods *Calculate item response probabilities*

Description

calcProb is a function for calculating item response probabilities.

Usage

```
calcProb(object, theta)

## S4 method for signature 'item_1PL,numeric'
calcProb(object, theta)

## S4 method for signature 'item_2PL,numeric'
calcProb(object, theta)

## S4 method for signature 'item_3PL,numeric'
calcProb(object, theta)

## S4 method for signature 'item_PC,numeric'
calcProb(object, theta)

## S4 method for signature 'item_GPC,numeric'
calcProb(object, theta)

## S4 method for signature 'item_GR,numeric'
calcProb(object, theta)

## S4 method for signature 'item_pool,numeric'
calcProb(object, theta)

## S4 method for signature 'item_1PL,matrix'
calcProb(object, theta)

## S4 method for signature 'item_2PL,matrix'
calcProb(object, theta)

## S4 method for signature 'item_3PL,matrix'
calcProb(object, theta)

## S4 method for signature 'item_PC,matrix'
calcProb(object, theta)

## S4 method for signature 'item_GPC,matrix'
calcProb(object, theta)

## S4 method for signature 'item_GR,matrix'
calcProb(object, theta)

## S4 method for signature 'item_pool,matrix'
calcProb(object, theta)

## S4 method for signature 'item_pool_cluster,numeric'
calcProb(object, theta)
```

Arguments

`object`	an `item` or an `item_pool` object.
`theta`	theta values to use.

Value

`item` **object:** `calcProb` returns a (*nq*, *ncat*) matrix of probability values.

`item_pool` **object:** `calcProb` returns a length *ni* list, each containing a matrix of probability values.

notations

- *nq* denotes the number of theta values.
- *ncat* denotes the number of response categories.
- *ni* denotes the number of items in the `item_pool` object.

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17.*

Examples

```
item_1      <- new("item_1PL", difficulty = 0.5)
item_2      <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3      <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4      <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5      <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
```

```
item_6      <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

prob_item_1 <- calcProb(item_1, seq(-3, 3, 1))
prob_item_2 <- calcProb(item_2, seq(-3, 3, 1))
prob_item_3 <- calcProb(item_3, seq(-3, 3, 1))
prob_item_4 <- calcProb(item_4, seq(-3, 3, 1))
prob_item_5 <- calcProb(item_5, seq(-3, 3, 1))
prob_item_6 <- calcProb(item_6, seq(-3, 3, 1))
prob_pool   <- calcProb(itempool_science, seq(-3, 3, 1))
```

`calculateAdaptivityMeasures` *Calculate Adaptivity Measures*

Description

`calculateAdaptivityMeasures` is a function for calculating commonly used adaptivity measures.

Usage

```
calculateAdaptivityMeasures(x, with_constraints = FALSE, config = NULL)
```

Arguments

`x`	an `output_Shadow_all` object.
`with_constraints`	`TRUE` to calculate best info with content constraints.
`config`	only used when `with_constraints` is `TRUE`. A `config_Static` object for calculating best information while satisfying content constraints. Leave this to `NULL` to use default settings of `createStaticTestConfig`.

Value

`calculateAdaptivityMeasures` returns a named list:

- `corr` the correlation between final theta estimates and average test locations.
- `ratio` the ratio of (1) standard deviation of average test locations, versus (2) standard deviation of final theta estimates.
- `PRV` the proportion of variance reduced, from (1) the variance of item locations of all items in the pool, by (2) the average of test location variances.
- `info` (1) average information of a test at final theta estimate, relative to (2) best average obtainable from item pool using same test length, adjusting for (3) average information from item pool using random selection.

checkConstraints *Check the consistency of constraints and item usage*

Description

Check the consistency of constraints and item usage.

Usage

```
checkConstraints(constraints, usage_matrix, true_theta = NULL)
```

Arguments

constraints	A constraints object generated by loadConstraints.
usage_matrix	A matrix of item usage data from Shadow.
true_theta	A vector of true theta values.

config_Shadow-class *Create a config_Shadow object*

Description

createShadowTestConfig is a config function for creating a config_Shadow object for shadowtest assembly. Default values are used for any unspecified parameters/slots.

Usage

```
createShadowTestConfig(
  item_selection = NULL,
  content_balancing = NULL,
  MIP = NULL,
  MCMC = NULL,
  exclude_policy = NULL,
  refresh_policy = NULL,
  exposure_control = NULL,
  overlap_control = NULL,
  stopping_criterion = NULL,
  interim_theta = NULL,
  final_theta = NULL,
  theta_grid = seq(-4, 4, 0.1)
)
```

Arguments

`item_selection` a named list containing item selection criteria.

- `method` the type of selection criteria. Accepts `MFI`, `MPWI`, `FB`, `EB`, `GFI`. (default = `MFI`)
- `info_type` the type of information. Accepts `FISHER`. (default = `FISHER`)
- `initial_theta` (optional) initial theta values to use.
- `fixed_theta` (optional) fixed theta values to use throughout all item positions.
- `target_value` (optional) the target value to use for `method = 'GFI'`.

`content_balancing` a named list containing content balancing options.

- `method` the type of balancing method. Accepts `NONE`, `STA`. (default = `STA`)

`MIP` a named list containing solver options.

- `solver` the type of solver. Accepts `Rsymphony`, `highs`, `gurobi`, `lpSolve`, `Rglpk`. (default = `LPSOLVE`)
- `verbosity` verbosity level of the solver. (default = `-2`)
- `time_limit` time limit in seconds. Used in solvers `Rsymphony`, `gurobi`, `Rglpk`. (default = `60`)
- `gap_limit` search termination criterion. Gap limit in relative scale passed onto the solver. Used in solver `gurobi`. (default = `.05`)
- `gap_limit_abs` search termination criterion. Gap limit in absolute scale passed onto the solver. Used in solvers `Rsymphony`. (default = `0.05`)
- `obj_tol` search termination criterion. The lower bound to use on the minimax deviation variable. Used when `item_selection$method` is `GFI`, and ignored otherwise. (default = `0.05`)
- `retry` number of times to retry running the solver if the solver returns no solution. Some solvers incorrectly return no solution even when a solution exists. This is the number of attempts to verify that the problem is indeed infeasible in such cases. Set to `0` to not retry. (default = `5`)

`MCMC` a named list containing Markov-chain Monte Carlo configurations for obtaining posterior samples.

- `burn_in` the number of chains from the start to discard. (default = `100`)
- `post_burn_in` the number of chains to use after discarding the first `burn_in` chains. (default = `500`)
- `thin` thinning interval to apply. `1` represents no thinning. (default = `1`)
- `jump_factor` the jump (scaling) factor for the proposal distribution. `1` represents no jumping. (default = `2.4`)

`exclude_policy` a named list containing the exclude policy for use with the `exclude` argument in Shadow.

- `method` the type of policy. Accepts `HARD`, `SOFT`. (default = `HARD`)
- `M` the Big M penalty to use on item information. Used in the `SOFT` method.

`refresh_policy` a named list containing the refresh policy for when to obtain a new shadowtest.

- method the type of policy. Accepts ALWAYS, POSITION, INTERVAL, THRESHOLD, INTERVAL-THRESHOLD, STIMULUS, SET, PASSAGE. (default = ALWAYS)
- interval used in methods INTERVAL, INTERVAL-THRESHOLD. Set to 1 to refresh at each position, 2 to refresh at every two positions, and so on. (default = 1)
- threshold used in methods THRESHOLD, INTERVAL-THRESHOLD. The absolute change in between interim theta estimates to trigger the refresh. (default = 0.1)
- position used in methods POSITION. Item positions to trigger the refresh. (default = 1)

exposure_control

a named list containing exposure control settings.

- method the type of exposure control method. Accepts NONE, ELIGIBILITY, BIGM, BIGM-BAYESIAN. (default = ELIGIBILITY)
- M used in methods BIGM, BIGM-BAYESIAN. the Big M penalty to use on item information.
- max_exposure_rate target exposure rates for each segment. (default = rep(0.25, 7))
- acceleration_factor the acceleration factor to apply. (default = 1)
- n_segment the number of theta segments to use. (default = 7)
- first_segment (optional) the theta segment assumed at the beginning of test for all participants.
- segment_cut theta segment cuts. (default = c(-Inf, seq(-2.5, 2.5, 1), Inf))
- initial_eligibility_stats (optional) initial eligibility statistics to use.
- fading_factor the fading factor to apply. (default = .999)
- diagnostic_stats set to TRUE to generate segment-wise diagnostic statistics. (default = FALSE)

overlap_control

a named list containing overlap control settings.

- method the type of overlap control method. Accepts NONE, ELIGIBILITY, BIGM, BIGM-BAYESIAN. (default = NONE)
- M used in methods BIGM, BIGM-BAYESIAN. the Big M penalty to use on item information.
- max_overlap_rate target overlap rate. (default = 0.20)

stopping_criterion

a named list containing stopping criterion.

- method the type of stopping criterion. Accepts FIXED. (default = FIXED)
- test_length test length.
- min_ni the maximum number of items to administer.
- max_ni the minimum number of items to administer.
- se_threshold standard error threshold. Item administration is stopped when theta estimate standard error becomes lower than this value.

interim_theta a named list containing interim theta estimation options.

- method the type of estimation. Accepts EAP, MLE, MLEF, EB, FB, CARRYOVER. (default = EAP)
- shrinkage_correction set TRUE to apply shrinkage correction. Used when method is EAP. (default = FALSE)
- prior_dist the type of prior distribution. Accepts NORMAL, UNIFORM. (default = NORMAL)
- prior_par distribution parameters for prior_dist. (default = c(0, 1))
- bound_ML theta bound in c(lower_bound, upper_bound) format. Used when method is MLE. (default = -4, 4)
- truncate_ML set TRUE to truncate ML estimate within bound_ML. (default = FALSE)
- max_iter maximum number of Newton-Raphson iterations. Used when method is MLE. (default = 50)
- crit convergence criterion. Used when method is MLE. (default = 1e-03)
- max_change maximum change in ML estimates between iterations. Changes exceeding this value is clipped to this value. Used when method is MLE. (default = 1.0)
- use_step_size set TRUE to use step_size. Used when method is MLE or MLEF. (default = FALSE)
- step_size upper bound to impose on the absolute change in initial theta and estimated theta. Absolute changes exceeding this value will be capped to step_size. Used when method is MLE or MLEF. (default = 0.5)
- do_Fisher set TRUE to use Fisher's method of scoring. Used when method is MLE. (default = TRUE)
- fence_slope slope parameter to use for method = 'MLEF'. This must have two values in total, for the lower and upper bound item respectively. Use one value to use the same value for both bounds. (default = 5)
- fence_difficulty difficulty parameters to use for method = 'MLEF'. This must have two values in total, for the lower and upper bound item respectively. (default = c(-5, 5))
- hand_scored_attribute (optional) the item attribute name for whether each item is hand-scored or not. The attribute should have TRUE (hand-scored) and FALSE (machine-scored) values. If a hand-scored item is administered to an examinee, the previous interim theta (or the starting theta if this occurs for the first item) is reused without updating the estimate.

final_theta a named list containing final theta estimation options.

- method the type of estimation. Accepts EAP, MLE, MLEF, EB, FB, CARRYOVER. (default = EAP)
- shrinkage_correction set TRUE to apply shrinkage correction. Used when method is EAP. (default = FALSE)
- prior_dist the type of prior distribution. Accepts NORMAL, UNIFORM. (default = NORMAL)
- prior_par distribution parameters for prior_dist. (default = c(0, 1))
- bound_ML theta bound in c(lower_bound, upper_bound) format. Used when method is MLE. (default = -4, 4)

- truncate_ML set TRUE to truncate ML estimate within bound_ML. (default = FALSE)
- max_iter maximum number of Newton-Raphson iterations. Used when method is MLE. (default = 50)
- crit convergence criterion. Used when method is MLE. (default = 1e-03)
- max_change maximum change in ML estimates between iterations. Changes exceeding this value is clipped to this value. Used when method is MLE. (default = 1.0)
- use_step_size set TRUE to use step_size. Used when method is MLE or MLEF. (default = FALSE)
- step_size upper bound to impose on the absolute change in initial theta and estimated theta. Absolute changes exceeding this value will be capped to step_size. Used when method is MLE or MLEF. (default = 0.5)
- do_Fisher set TRUE to use Fisher's method of scoring. Used when method is MLE. (default = TRUE)
- fence_slope slope parameter to use for method = 'MLEF'. This must have two values in total, for the lower and upper bound item respectively. Use one value to use the same value for both bounds. (default = 5)
- fence_difficulty difficulty parameters to use for method = 'MLEF'. This must have two values in total, for the lower and upper bound item respectively. (default = c(-5, 5))

theta_grid the theta grid to use as quadrature points.

Examples

```
cfg1 <- createShadowTestConfig(refresh_policy = list(
  method = "STIMULUS"
))
cfg2 <- createShadowTestConfig(refresh_policy = list(
  method = "POSITION",
  position = c(1, 5, 9)
))
```

---

config_Static-class *Create a config_Static object*

Description

createStaticTestConfig is a config function for creating a config_Static object for Static (fixed-form) test assembly. Default values are used for any unspecified parameters/slots.

Usage

```
createStaticTestConfig(item_selection = NULL, MIP = NULL)
```

Arguments

`item_selection` a named list containing item selection criteria.

- `method` the type of selection criteria. Accepts `MAXINFO`, `TIF`, `TCC`. (default = `MAXINFO`)
- `info_type` the type of information. Accepts `FISHER`. (default = `FISHER`)
- `target_location` a numeric vector containing the locations of target theta points. (e.g. `c(-1, 0, 1)`) (default = `c(-1.2, 0, 1.2)`)
- `target_value` a numeric vector containing the target values at each theta location. This should have the same length with `target_location`. Ignored if method is `MAXINFO`. (default = `NULL`)
- `target_weight` a numeric vector containing the weights for each theta location. This should have the same length with `target_location`. (default = `rep(1, length(target_location))`

`MIP` a named list containing solver options.

- `solver` the type of solver. Accepts `Rsymphony`, `highs`, `gurobi`, `lpSolve`, `Rglpk`. (default = `LPSOLVE`)
- `verbosity` verbosity level of the solver. (default = `-2`)
- `time_limit` time limit in seconds. Used in solvers `Rsymphony`, `gurobi`, `Rglpk`. (default = `60`)
- `gap_limit` search termination criterion. Gap limit in relative scale passed onto the solver. Used in solver `gurobi`. (default = `.05`)
- `gap_limit_abs` search termination criterion. Gap limit in absolute scale passed onto the solver. Used in solvers `Rsymphony`. (default = `0.05`)
- `obj_tol` search termination criterion. The lower bound to use on the minimax deviation variable. Used when `item_selection$method` is `TIF` or `TCC`. (default = `0.05`)
- `retry` number of times to retry running the solver if the solver returns no solution. Some solvers incorrectly return no solution even when a solution exists. This is the number of attempts to verify that the problem is indeed infeasible in such cases. Set to `0` to not retry. (default = `5`)

Value

`createStaticTestConfig` returns a `config_Static` object. This object is used in `Static`.

Examples

```
cfg1 <- createStaticTestConfig(
  list(
    method = "MAXINFO",
    info_type = "FISHER",
    target_location = c(-1, 0, 1),
    target_weight = c(1, 1, 1)
  )
)

cfg2 <- createStaticTestConfig(
```

```
  list(
    method = "TIF",
    info_type = "FISHER",
    target_location = c(-1, 0, 1),
    target_weight = c(1, 1, 1),
    target_value = c(8, 10, 12)
  )
)

cfg3 <- createStaticTestConfig(
  list(
    method = "TCC",
    info_type = "FISHER",
    target_location = c(-1, 0, 1),
    target_weight = c(1, 1, 1),
    target_value = c(10, 15, 20)
  )
)
```

`constraint-class` *Class 'constraint': a single constraint*

Description

`constraint` is an S4 class for representing a single constraint.

Slots

`constraint` the numeric index of the constraint.

`constraint_id` the character ID of the constraint.

`nc` the number of MIP-format constraints translated from this constraint.

`mat,dir,rhs` these represent MIP-format constraints. A single MIP-format constraint is associated with a row in `mat`, a value in `rhs`, and a value in `dir`.

- the *i*-th row of `mat` represents LHS coefficients to use on decision variables in the *i*-th MIP-format constraint.
- the *i*-th value of `rhs` represents RHS values to use in the *i*-th MIP-format constraint.
- the *i*-th value of `dir` represents the imposed constraint between LHS and RHS.

`suspend` `TRUE` if the constraint is not to be imposed.

constraints-class *Class 'constraints': a set of constraints*

Description

constraints is an S4 class for representing a set of constraints and its associated objects.

Details

See constraints-operators for object manipulation functions.

Slots

constraints a data.frame containing the constraint specifications.

list_constraints a list containing the constraint object representation of each constraint.

pool the item_pool object associated with the constraints.

item_attrib the item_attrib object associated with the constraints.

st_attrib the st_attrib object associated with the constraints.

test_length the test length specified in the constraints.

nv the number of decision variables. Equals ni + ns.

ni the number of items to search from.

ns the number of stimulus to search from.

id the item/stimulus ID string of each item/stimulus.

index,mat,dir,rhs these represent MIP-format constraints. A single MIP-format constraint is associated with a value in index, a row in mat, a value in rhs, and a value in dir.

- the *i*-th value of index represents which constraint specification in the constraints argument it was translated from.
- the *i*-th row of mat represents LHS coefficients to use on decision variables in the *i*-th MIP-format constraint.
- the *i*-th value of rhs represents RHS values to use in the *i*-th MIP-format constraint.
- the *i*-th value of dir represents the imposed constraint between LHS and RHS.

set_based TRUE if the constraint is set-based. FALSE otherwise.

item_order the item attribute of each item to use in imposing an item order constraint, if any.

item_order_by the name of the item attribute to use in imposing an item order constraint, if any.

stim_order the stimulus attribute of each stimulus to use in imposing a stimulus order constraint, if any.

stim_order_by the name of the stimulus attribute to use in imposing a stimulus order constraint, if any.

item_index_by_stimulus a list containing item indices of each stimulus.

stimulus_index_by_item the stimulus indices of each item.

constraints-operators *Basic operators for constraints objects*

Description

Create a subset of a constraints object:

- constraints[i]
- subsetConstraints(constraints, 1:10)

Combine two constraints objects:

- c(constraints1, constraints2)
- combineConstraints(constraints1, constraints2)

Usage

```
subsetConstraints(x, i = NULL)

combineConstraints(x1, x2)

## S4 method for signature 'constraints,numeric'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'constraints'
c(x, ...)
```

Arguments

x, x1, x2	a constraints object.
i, j	indices to use in subsetting.
...	not used, exists for compatibility.
drop	not used, exists for compatibility.

Examples

```
c1 <- constraints_science
c2 <- c1[1:10]
c3 <- c1[c(1, 11:36)] # keep constraint 1 for test length
c4 <- c(c2, c3)
```

dataset_bayes *Bayes dataset*

Description

Item-based example item pool with standard errors (320 items).

Details

This pool is associated with the following objects:

- `itempool_bayes` an `item_pool` object containing 320 items.
- `itemattrib_bayes` a `item_attrib` object containing 5 item-level attributes.
- `constraints_bayes` a `constraints` object containing 14 constraints.

Also, the following objects are intended for illustrating expected data structures.

- `itempool_bayes_data` a `data.frame` containing item parameters.
- `itempool_se_bayes_data` a `data.frame` containing item parameter standard errors.
- `itemattrib_bayes_data` a `data.frame` containing item attributes.
- `constraints_bayes_data` a `data.frame` containing constraint specifications.

Examples

```
itempool_bayes    <- loadItemPool(itempool_bayes_data, itempool_se_bayes_data)
itemattrib_bayes  <- loadItemAttrib(itemattrib_bayes_data, itempool_bayes)
constraints_bayes <- loadConstraints(constraints_bayes_data,
  itempool_bayes, itemattrib_bayes)
```

dataset_fatigue *Fatigue dataset*

Description

Item-based example pool with item contents (95 items).

Details

This pool is associated with the following objects:

- `itempool_fatigue` an `item_pool` object containing 95 items.
- `itemattrib_fatigue` an `item_attrib` object containing 7 item-level attributes.
- `constraints_fatigue` a `constraints` object containing 111 constraints.

Also, the following objects are intended for illustrating expected data structures.

- `itempool_fatigue_data` a `data.frame` containing item parameters.
- `itemattrib_fatigue_data` a `data.frame` containing item attributes.
- `itemtext_fatigue_data` a `data.frame` containing item texts.
- `constraints_fatigue_data` a `data.frame` containing constraint specifications.
- `resp_fatigue_data` a `data.frame` containing raw response data.

Examples

```
itempool_fatigue   <- loadItemPool(itempool_fatigue_data)
itemattrib_fatigue <- loadItemAttrib(itemattrib_fatigue_data, itempool_fatigue)
constraints_fatigue <- loadConstraints(constraints_fatigue_data,
  itempool_fatigue, itemattrib_fatigue)
```

---

dataset_reading *Reading dataset*

Description

Stimulus-based example item pool (303 items, 35 stimuli).

Details

This pool is associated with the following objects:

- `itempool_reading` an `item_pool` object containing 303 items.
- `itemattrib_reading` an `item_attrib` object containing 12 item-level attributes.
- `stimattrib_reading` a `st_attrib` object containing 4 stimulus-level attributes.
- `constraints_reading` a `constraints` object containing 18 constraints.

Also, the following objects are intended for illustrating expected data structures.

- `itempool_reading_data` a `data.frame` containing item parameters.
- `itemattrib_reading_data` a `data.frame` containing item attributes.
- `stimattrib_reading_data` a `data.frame` containing stimulus attributes.
- `constraints_reading_data` a `data.frame` containing constraint specifications.

Examples

```
itempool_reading    <- loadItemPool(itempool_reading_data)
itemattrib_reading  <- loadItemAttrib(itemattrib_reading_data, itempool_reading)
stimattrib_reading  <- loadStAttrib(stimattrib_reading_data, itemattrib_reading)
constraints_reading <- loadConstraints(constraints_reading_data,
  itempool_reading, itemattrib_reading, stimattrib_reading)
```

dataset_science *Science dataset*

Description

Item-based example item pool (1000 items).

Details

This pool is associated with the following objects:

- `itempool_science` an `item_pool` object containing 1000 items.
- `itemattrib_science` an `item_attrib` object containing 9 item-level attributes.
- `constraints_science` a `constraints` object containing 36 constraints.

Also, the following objects are intended for illustrating expected data structures.

- `itempool_science_data` a `data.frame` containing item parameters.
- `itemattrib_science_data` a `data.frame` containing item attributes.
- `constraints_science_data` a `data.frame` containing constraint specifications.

Examples

```
itempool_science    <- loadItemPool(itempool_science_data)
itemattrib_science  <- loadItemAttrib(itemattrib_science_data, itempool_science)
constraints_science <- loadConstraints(constraints_science_data,
  itempool_science, itemattrib_science)
```

detectBestSolver *Detect best solver*

Description

Detect best solver

Usage

```
detectBestSolver()
```

Value

the package name of the best available solver on the system.

Examples

```
solver <- detectBestSolver()
cfg <- createStaticTestConfig(MIP = list(solver = solver))
cfg <- createShadowTestConfig(MIP = list(solver = solver))
```

`e_item` *(C++) Calculate expected scores*

Description

`e_*()` and `array_e_*()` are C++ functions for calculating expected scores.

Usage

```
e_1pl(x, b)

e_2pl(x, a, b)

e_m_2pl(x, a, d)

e_3pl(x, a, b, c)

e_m_3pl(x, a, d, c)

e_pc(x, b)

e_gpc(x, a, b)

e_m_gpc(x, a, d)

e_gr(x, a, b)

e_m_gr(x, a, d)

array_e_1pl(x, b)

array_e_2pl(x, a, b)

array_e_3pl(x, a, b, c)

array_e_pc(x, b)

array_e_gpc(x, a, b)

array_e_gr(x, a, b)
```

Arguments

x	the theta value. The number of columns should correspond to the number of dimensions. For `array_*()` functions, the number of theta values must correspond to the number of rows.
b, d	the difficulty parameter. `b` is used for unidimensional items, and `d` is used for multidimensional items.
a	the *a*-parameter.
c	the *c*-parameter.

Details

`e_*()` functions accept a single theta value, and `array_p_*()` functions accept multiple theta values.

Supports unidimensional and multidimensional models.

- `e_1pl()`, `array_e_1pl()`: 1PL models
- `e_2pl()`, `array_e_2pl()`: 2PL models
- `e_3pl()`, `array_e_3pl()`: 3PL models
- `e_pc()`, `array_e_pc()`: PC (partial credit) models
- `e_gpc()`, `array_e_gpc()`: GPC (generalized partial credit) models
- `e_gr()`, `array_e_gr()`: GR (graded response) models
- `e_m_2pl()`, `array_e_m_2pl()`: multidimensional 2PL models
- `e_m_3pl()`, `array_e_m_3pl()`: multidimensional 3PL models
- `e_m_gpc()`, `array_e_m_gpc()`: multidimensional GPC models
- `e_m_gr()`, `array_e_m_gr()`: multidimensional GR models

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
x <- 0.5

e_1pl(x, 1)
e_2pl(x, 1, 2)
e_3pl(x, 1, 2, 0.25)
e_pc(x, c(0, 1))
e_gpc(x, 2, c(0, 1))
e_gr(x, 2, c(0, 2))

x <- matrix(seq(-3, 3, 1)) # three theta values, unidimensional

array_e_1pl(x, 1)
array_e_2pl(x, 1, 2)
array_e_3pl(x, 1, 2, 0.25)
array_e_pc(x, c(0, 1))
array_e_gpc(x, 2, c(0, 1))
array_e_gr(x, 2, c(0, 2))
```

---

eap *Compute expected a posteriori estimates of theta*

Description

eap is a function for computing expected a posteriori estimates of theta.

Usage

```
eap(
  object,
  select = NULL,
  resp,
  theta_grid = seq(-4, 4, 0.1),
  prior = rep(1/81, 81)
)

## S4 method for signature 'item_pool'
eap(
  object,
```

```
  select = NULL,
  resp,
  theta_grid = seq(-4, 4, 0.1),
  prior = rep(1/81, 81)
)

EAP(object, select = NULL, prior, reset_prior = FALSE)

## S4 method for signature 'test'
EAP(object, select = NULL, prior, reset_prior = FALSE)

## S4 method for signature 'test_cluster'
EAP(object, select = NULL, prior, reset_prior = FALSE)
```

Arguments

object	an item_pool object.
select	(optional) if item indices are supplied, only the specified items are used.
resp	item response on all (or selected) items in the object argument. Can be a vector, a matrix, or a data frame. length(resp) or ncol(resp) must be equal to the number of all (or selected) items.
theta_grid	the theta grid to use as quadrature points. (default = seq(-4, 4, .1))
prior	a prior distribution, a numeric vector for a common prior or a matrix for individualized priors. (default = rep(1 / 81, 81))
reset_prior	used for test_cluster objects. If TRUE, reset the prior distribution for each test object.

Value

eap returns a list containing estimated values.

- th theta value.
- se standard error.

Examples

```
eap(itempool_fatigue, resp = resp_fatigue_data[10, ])
eap(itempool_fatigue, select = 1:20, resp = resp_fatigue_data[10, 1:20])
```

find_segment *(C++) Classify theta values into segments using cutpoints*

Description

`find_segment()` is a function for classifying theta values into segments based on supplied cutpoints.

Usage

```
find_segment(x, segment)
```

Arguments

`x`	the theta value. This can be a vector.
`segment`	segment cutpoints. Values of `-Inf`, `Inf` are not implied and must be explicitly supplied if intended.

Examples

```
cuts <- c(-Inf, -2, 0, 2, Inf)

find_segment(-3, cuts)
find_segment(-1, cuts)
find_segment(1, cuts)
find_segment(3, cuts)
find_segment(seq(-3, 3, 2), cuts)
```

getScoreAttributes *Retrieve constraints-related scores from solution*

Description

`getScoreAttributes` is a helper function for retrieving constraints-related scores from a solution.

Usage

```
getScoreAttributes(constraints, item_idx, item_resp, item_ncat)
```

Arguments

`constraints`	a `constraints` object.
`item_idx`	item indices from a solution.
`item_resp`	item scores for `item_idx`.
`item_ncat`	number of score categories for `item_idx`.

Examples

```
item_idx <-
  c( 29,  33,  26,  36,  34,
    295, 289, 296, 291, 126,
    133, 124, 134, 129,  38,
     47,  39,  41,  46,  45,
    167, 166, 170, 168, 113,
    116, 119, 117, 118, 114)

item_resp <-
  c( 1, 0, 1, 1, 0,
     0, 1, 1, 0, 0,
     1, 0, 1, 0, 1,
     1, 1, 1, 0, 1,
     0, 1, 1, 1, 1,
     1, 0, 1, 0, 1)

item_ncat <-
  c( 2, 2, 2, 2, 2,
     2, 2, 2, 2, 2,
     2, 2, 2, 2, 2,
     2, 2, 2, 2, 2,
     2, 2, 2, 2, 2,
     2, 2, 2, 2, 2)

getScoreAttributes(constraints_reading, item_idx, item_resp, item_ncat)
```

---

getSolution *Print solution items*

Description

Print solution items

Usage

```
getSolution(object, examinee = NA, position = NA, index_only = TRUE)

## S4 method for signature 'list'
getSolution(object, examinee = NA, position = NA, index_only = TRUE)

## S4 method for signature 'output_Static'
getSolution(object, examinee = NA, position = NA, index_only = TRUE)
```

Arguments

object	an output_Static object or an output_Shadow object.

examinee	(optional) the examinee index to display the solution. Used when the 'object' argument is an output_Shadow object.
position	(optional) if supplied, display the item attributes of the assembled test at that item position. If not supplied, display the item attributes of the administered items. Used when the 'object' argument is an output_Shadow object.
index_only	if TRUE, only print item indices. if FALSE, print all item attributes. (default = TRUE)

Value

Item attributes of solution items.

---

getSolutionAttributes *Retrieve constraints-related attributes from solution*

Description

getSolutionAttributes is a helper function for retrieving constraints-related attributes from a solution.

Usage

```
getSolutionAttributes(constraints, item_idx, all_values = FALSE)
```

Arguments

constraints	a constraints object.
item_idx	item indices from a solution.
all_values	if TRUE, return all values as-is without taking the mean when there are multiple values. If FALSE, return the mean when there are multiple values. This has an effect when there is a constraint on items per stimulus, where there are multiple values of number of items per stimulus. In this case, if TRUE, the number of items for every stimuli are returned as-is. If FALSE, the average number of items across stimuli is returned. (default = FALSE)

Value

- If all_values == FALSE, getSolutionAttributes returns a data.frame containing constraints data and their associated attributes.
- If all_values == TRUE, getSolutionAttributes returns a list containing attributes associated to each constraint.

Examples

```
item_idx <-
  c( 29,  33,  26,  36,  34,
    295, 289, 296, 291, 126,
    133, 124, 134, 129,  38,
     47,  39,  41,  46,  45,
    167, 166, 170, 168, 113,
    116, 119, 117, 118, 114)

getSolutionAttributes(constraints_reading, item_idx, FALSE)
getSolutionAttributes(constraints_reading, item_idx, TRUE)
```

---

h_item *(C++) Calculate second derivative of log-likelihood*

Description

`h_*()` and `array_h_*()` are C++ functions for calculating the second derivative of the log-likelihood function.

Usage

```
h_1pl(x, b, u)

h_2pl(x, a, b, u)

h_m_2pl(x, a, d, u)

h_3pl(x, a, b, c, u)

h_m_3pl(x, a, d, c, u)

h_pc(x, b, u)

h_gpc(x, a, b, u)

h_m_gpc(x, a, d, u)

h_gr(x, a, b, u)

h_m_gr(x, a, d, u)

array_h_1pl(x, b, u)

array_h_2pl(x, a, b, u)
```

```
array_h_3pl(x, a, b, c, u)

array_h_pc(x, b, u)

array_h_gpc(x, a, b, u)

array_h_gr(x, a, b, u)
```

Arguments

x	the theta value. The number of columns should correspond to the number of dimensions. For `array_*()` functions, the number of theta values must correspond to the number of rows.
b, d	the difficulty parameter. `b` is used for unidimensional items, and `d` is used for multidimensional items.
u	the response value.
a	the *a*-parameter.
c	the *c*-parameter.

Details

`h_*()` functions accept a single theta value, and `array_h_*()` functions accept multiple theta values.

Supports unidimensional and multidimensional models.

- `h_1pl()`, `array_h_1pl()`: 1PL models
- `h_2pl()`, `array_h_2pl()`: 2PL models
- `h_3pl()`, `array_h_3pl()`: 3PL models
- `h_pc()`, `array_h_pc()`: PC (partial credit) models
- `h_gpc()`, `array_h_gpc()`: GPC (generalized partial credit) models
- `h_gr()`, `array_h_gr()`: GR (graded response) models
- `h_m_2pl()`, `array_h_m_2pl()`: multidimensional 2PL models
- `h_m_3pl()`, `array_h_m_3pl()`: multidimensional 3PL models
- `h_m_gpc()`, `array_h_m_gpc()`: multidimensional GPC models
- `h_m_gr()`, `array_h_m_gr()`: multidimensional GR models

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
u <- 1

x <- 0.5
h_1pl(x, 1, u)
h_2pl(x, 1, 2, u)
h_3pl(x, 1, 2, 0.25, u)
h_pc(x, c(0, 1), u)
h_gpc(x, 2, c(0, 1), u)
h_gr(x, 2, c(0, 2), u)

x <- matrix(seq(-3, 3, 1)) # three theta values, unidimensional
array_h_1pl(x, 1, u)
array_h_2pl(x, 1, 2, u)
array_h_3pl(x, 1, 2, 0.25, u)
array_h_pc(x, c(0, 1), u)
array_h_gpc(x, 2, c(0, 1), u)
array_h_gr(x, 2, c(0, 2), u)
```

`info_item` *(C++) Calculate Fisher information*

Description

`info_*()` and `array_info_*()` are functions for calculating Fisher information.

Usage

```
info_1pl(x, b)

info_2pl(x, a, b)

info_m_2pl(x, a, d)

dirinfo_m_2pl(x, a, d)

thisdirinfo_m_2pl(x, alpha_vec, a, d)

info_3pl(x, a, b, c)

info_m_3pl(x, a, d, c)

dirinfo_m_3pl(x, a, d, c)

thisdirinfo_m_3pl(x, alpha_vec, a, d, c)

info_pc(x, b)

info_gpc(x, a, b)

info_m_gpc(x, a, d)

dirinfo_m_gpc(x, a, d)

thisdirinfo_m_gpc(x, alpha_vec, a, d)

info_gr(x, a, b)

info_m_gr(x, a, d)

dirinfo_m_gr(x, a, d)

thisdirinfo_m_gr(x, alpha_vec, a, d)

array_info_1pl(x, b)

array_info_2pl(x, a, b)

array_info_m_2pl(x, a, d)

array_dirinfo_m_2pl(x, a, d)

array_thisdirinfo_m_2pl(x, alpha_vec, a, d)

array_info_3pl(x, a, b, c)
```

```
array_info_m_3pl(x, a, d, c)

array_dirinfo_m_3pl(x, a, d, c)

array_thisdirinfo_m_3pl(x, alpha_vec, a, d, c)

array_info_pc(x, b)

array_info_gpc(x, a, b)

array_info_m_gpc(x, a, d)

array_dirinfo_m_gpc(x, a, d)

array_thisdirinfo_m_gpc(x, alpha_vec, a, d)

array_info_gr(x, a, b)

array_info_m_gr(x, a, d)

array_dirinfo_m_gr(x, a, d)

array_thisdirinfo_m_gr(x, alpha_vec, a, d)
```

Arguments

x	the theta value. The number of columns should correspond to the number of dimensions. For `array_*()` functions, the number of theta values must correspond to the number of rows.
b, d	the difficulty parameter. `b` is used for unidimensional items, and `d` is used for multidimensional items.
a	the *a*-parameter.
alpha_vec	the alpha angle vector. Used for directional information in `thisdirinfo_*()` and `array_thisdirinfo_*()`.
c	the *c*-parameter.

Details

`info_*()` functions accept a single theta value, and `array_info_*` functions accept multiple theta values.

Supports unidimensional and multidimensional models.

- `info_1pl()`, `array_info_1pl()`: 1PL models
- `info_2pl()`, `array_info_2pl()`: 2PL models
- `info_3pl()`, `array_info_3pl()`: 3PL models
- `info_pc()`, `array_info_pc()`: PC (partial credit) models

- `info_gpc()`, `array_info_gpc()`: GPC (generalized partial credit) models
- `info_gr()`, `array_info_gr()`: GR (graded response) models
- `info_m_2pl()`, `array_info_m_2pl()`: multidimensional 2PL models
- `info_m_3pl()`, `array_info_m_3pl()`: multidimensional 3PL models
- `info_m_gpc()`, `array_info_m_gpc()`: multidimensional GPC models
- `info_m_gr()`, `array_info_m_gr()`: multidimensional GR models
- Directional information for a specific angle
  - `thisdirinfo_m_2pl()`, `array_thisdirinfo_m_2pl()`: multidimensional 2PL models
  - `thisdirinfo_m_3pl()`, `array_thisdirinfo_m_3pl()`: multidimensional 3PL models
  - `thisdirinfo_m_gpc()`, `array_thisdirinfo_m_gpc()`: multidimensional GPC models
  - `thisdirinfo_m_gr()`, `array_thisdirinfo_m_gr()`: multidimensional GR models

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
x <- 0.5

info_1pl(x, 1)
info_2pl(x, 1, 2)
info_3pl(x, 1, 2, 0.25)
info_pc(x, c(0, 1))
```

```
info_gpc(x, 2, c(0, 1))
info_gr(x, 2, c(0, 2))

x <- matrix(seq(0.1, 0.5, 0.1)) # three theta values, unidimensional

array_info_1pl(x, 1)
array_info_2pl(x, 1, 2)
array_info_3pl(x, 1, 2, 0.25)
array_info_pc(x, c(0, 1))
array_info_gpc(x, 2, c(0, 1))
array_info_gr(x, 2, c(0, 2))
```

`iparPosteriorSample` *Generate item parameter samples for Bayesian purposes*

Description

`iparPosteriorSample` is a function for generating item parameter samples. Used for the FB (full-Bayesian) estimation method.

Usage

```
iparPosteriorSample(pool, n_sample = 500)
```

Arguments

`pool`	an `item_pool` object.
`n_sample`	the number of samples to draw.

Value

`iparPosteriorSample` returns a length-*ni* list of item parameter matrices, with each matrix having `n_sample` rows.

Examples

```
ipar <- iparPosteriorSample(itempool_bayes, 5)
ipar <- iparPosteriorSample(itempool_science, 5) # no variation
```

`item-classes` *Item classes*

Description

- `item_1PL` class represents a 1PL item.
- `item_2PL` class represents a 2PL item.
- `item_3PL` class represents a 3PL item.
- `item_PC` class represents a partial credit item.
- `item_GPC` class represents a generalized partial credit item.
- `item_GR` class represents a graded response item.

Slots

`slope` a slope parameter value

`difficulty` a difficulty parameter value

`guessing` a guessing parameter value

`threshold` a vector of threshold parameter values

`category` a vector of category boundary values

`ncat` the number of response categories

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
item_1 <- new("item_1PL", difficulty = 0.5)
item_2 <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3 <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4 <- new("item_PC", threshold = c(-0.5, 0.5), ncat = 3)
item_5 <- new("item_GPC", slope = 1.0, threshold = c(-0.5, 0.0, 0.5), ncat = 4)
item_6 <- new("item_GR", slope = 1.0, category = c(-2.0, -1.0, 0, 1.0, 2.0), ncat = 6)
```

---

item_attrib-class *Load item attributes*

Description

loadItemAttrib is a data loading function for creating an item_attrib object. loadItemAttrib can read item attributes from a data.frame or a .csv file.

Usage

```
loadItemAttrib(object, pool)
```

Arguments

object	item attributes. Can be a data.frame or the file path of a .csv file. The content should at least include an 'ID' column that matches with item IDs (the 'ID' column) of the item_pool object.
pool	an item_pool object. Use loadItemPool for this.

Value

loadItemAttrib returns an item_attrib object.

- data a data.frame containing item attributes.

See Also

dataset_science, dataset_reading, dataset_fatigue, dataset_bayes for examples.

Examples

```
## Read from data.frame:
itempool_science   <- loadItemPool(itempool_science_data)
itemattrib_science <- loadItemAttrib(itemattrib_science_data, itempool_science)

## Read from file: write to tempdir() for illustration and clean afterwards
f <- file.path(tempdir(), "itemattrib_science.csv")
write.csv(itemattrib_science_data, f, row.names = FALSE)
itemattrib_science <- loadItemAttrib(f, itempool_science)
file.remove(f)
```

`item_attrib-operators` *Basic functions for item attribute objects*

Description

Basic functions for item attribute objects

Usage

```
## S4 method for signature 'item_attrib,numeric'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'item_attrib'
dim(x)

## S4 method for signature 'item_attrib'
colnames(x)

## S4 method for signature 'item_attrib'
rownames(x)

## S4 method for signature 'item_attrib'
names(x)

## S4 method for signature 'item_attrib'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	an `item_attrib` object.
`i, j`	indices to use in subsetting.
`...`	not used, exists for compatibility.
`drop`	not used, exists for compatibility.
`row.names`	not used, exists for compatibility.
`optional`	not used, exists for compatibility.

Examples

```
x <- itemattrib_science
x[1:10]
dim(x)
ncol(x)
nrow(x)
colnames(x)
rownames(x)
names(x)
```

```
as.data.frame(x)
```

`item_pool-class` *Class 'item_pool': an item pool*

Description

`item_pool` is an S4 class for representing an item pool.

Details

See `item_pool-operators` for object manipulation functions.

Slots

`ni` the number of items in the pool.
`max_cat` the maximum number of response categories across the pool.
`index` the numeric index of each item.
`id` the ID string of each item.
`model` the item class name of each item. See `item-classes`.
`NCAT` the number of response categories of each item.
`parms` a list containing item class objects. See `item-classes`.
`ipar` a matrix containing item parameters.
`se` a matrix containing item parameter standard errors.
`raw` the raw input `data.frame` used in `loadItemPool` to create this object.
`raw_se` the raw input `data.frame` used in `loadItemPool` to create this object.
`unique` whether item IDs must be unique for this object to be a valid object.

`item_pool-operators` *Basic operators for item pool objects*

Description

Create a subset of an `item_pool` object:

- `pool[i]`
- `subsetItemPool(pool, i)`

Combine two `item_pool` objects:

- `c(pool1, pool2)`
- `combineItemPool(pool1, pool2)`
- `pool1 + pool2`

`pool1 - pool2` excludes items in `pool2` from `pool1`.

`pool1 == pool2` tests whether two `item_pool` objects are identical.

Usage

```
subsetItemPool(x, i = NULL)

combineItemPool(x1, x2, unique = TRUE, verbose = TRUE)

## S4 method for signature 'item_pool,numeric'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'item_pool'
c(x, ...)

## S3 method for class 'item_pool'
x1 + x2

## S3 method for class 'item_pool'
x1 - x2

## S3 method for class 'item_pool'
x1 == x2
```

Arguments

x, x1, x2	an item_pool object.
i	item indices to use in subsetting.
unique	if TRUE, remove items with duplicate IDs after combining. (default = TRUE)
verbose	if TRUE, raise a warning if duplicate IDs are found after combining. (default = TRUE)
j, drop, ...	not used, exists for compatibility.

Examples

```
p1 <- itempool_science[1:100]
p2 <- c(itempool_science, itempool_reading)
p3 <- p2 - p1

p1 <- itempool_science[1:500]
p2 <- itempool_science - p1
p3 <- itempool_science[501:1000]
identical(p2, p3)  ## TRUE

p <- p1 + p3
p == itempool_science ## TRUE
```

item_pool_cluster-class *Class 'item_pool_cluster': an item pool*

Description

item_pool_cluster is an S4 class for representing a group of item pools.

Slots

np the number of item pools.

pools a list of item_pool objects.

names a vector containing item pool names.

j_item *(C++) Calculate first derivative of log-likelihood*

Description

j_*() and array_j_*() are C++ functions for calculating the first derivative of the log-likelihood function.

Usage

```
j_1pl(x, b, u)

j_2pl(x, a, b, u)

j_m_2pl(x, a, d, u)

j_3pl(x, a, b, c, u)

j_m_3pl(x, a, d, c, u)

j_pc(x, b, u)

j_gpc(x, a, b, u)

j_m_gpc(x, a, d, u)

j_gr(x, a, b, u)

j_m_gr(x, a, d, u)
```

```
array_j_1pl(x, b, u)

array_j_2pl(x, a, b, u)

array_j_3pl(x, a, b, c, u)

array_j_pc(x, b, u)

array_j_gpc(x, a, b, u)

array_j_gr(x, a, b, u)
```

Arguments

x	the theta value. The number of columns should correspond to the number of dimensions. For `array_*()` functions, the number of theta values must correspond to the number of rows.
b, d	the difficulty parameter. `b` is used for unidimensional items, and `d` is used for multidimensional items.
u	the response value.
a	the *a*-parameter.
c	the *c*-parameter.

Details

`j_*()` functions accept a single theta value, and `array_j_*()` functions accept multiple theta values.

Supports unidimensional and multidimensional models.

- `j_1pl()`, `array_j_1pl()`: 1PL models
- `j_2pl()`, `array_j_2pl()`: 2PL models
- `j_3pl()`, `array_j_3pl()`: 3PL models
- `j_pc()`, `array_j_pc()`: PC (partial credit) models
- `j_gpc()`, `array_j_gpc()`: GPC (generalized partial credit) models
- `j_gr()`, `array_j_gr()`: GR (graded response) models
- `j_m_2pl()`, `array_j_m_2pl()`: multidimensional 2PL models
- `j_m_3pl()`, `array_j_m_3pl()`: multidimensional 3PL models
- `j_m_gpc()`, `array_j_m_gpc()`: multidimensional GPC models
- `j_m_gr()`, `array_j_m_gr()`: multidimensional GR models

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
u <- 1

x <- 0.5
j_1pl(x, 1, u)
j_2pl(x, 1, 2, u)
j_3pl(x, 1, 2, 0.25, u)
j_pc(x, c(0, 1), u)
j_gpc(x, 2, c(0, 1), u)
j_gr(x, 2, c(0, 2), u)

x <- matrix(seq(-3, 3, 1)) # three theta values, unidimensional
array_j_1pl(x, 1, u)
array_j_2pl(x, 1, 2, u)
array_j_3pl(x, 1, 2, 0.25, u)
array_j_pc(x, c(0, 1), u)
array_j_gpc(x, 2, c(0, 1), u)
array_j_gr(x, 2, c(0, 2), u)
```

---

lnHyperPars *Convert mean and standard deviation into log-normal distribution parameters*

Description

lnHyperPars is a function for calculating parameters for a log-normal distribution, such that the distribution yields desired mean and standard deviation. Used for sampling the a-parameter.

Usage

```
lnHyperPars(mean, sd)
```

Arguments

mean	the desired mean.
sd	the desired standard deviation.

Value

lnHyperPars returns two values. These can be directly supplied to rlnorm.

Examples

```
pars <- lnHyperPars(2, 4)
x <- rlnorm(1000000, pars[1], pars[2])
mean(x) # close to 2
sd(x)   # close to 4
```

---

loadConstraints *Load constraints*

Description

loadConstraints is a data loading function for creating a constraints object. loadConstraints can read constraints from a data.frame or a .csv file. The contents must be in the expected format; see the vignette in vignette("constraints") for a documentation.

Usage

```
loadConstraints(object, pool, item_attrib, st_attrib = NULL)
```

Arguments

object	constraint specifications. Can be a data.frame or the file path of a .csv file. See the vignette for a description of the expected format.
pool	an item_pool object. Use loadItemPool for this.
item_attrib	an item_attrib object. Use loadItemAttrib for this.
st_attrib	(optional) an st_attrib object. Use loadStAttrib for this.

Value

loadConstraints returns a constraints object. This object is used in Static and Shadow.

See Also

dataset_science, dataset_reading, dataset_fatigue, dataset_bayes for examples.

Examples

```
## Read from data.frame:
itempool_science    <- loadItemPool(itempool_science_data)
itemattrib_science  <- loadItemAttrib(itemattrib_science_data, itempool_science)
constraints_science <- loadConstraints(constraints_science_data,
  itempool_science, itemattrib_science)

## Read from file: write to tempdir() for illustration and clean afterwards
f <- file.path(tempdir(), "constraints_science.csv")
write.csv(constraints_science_data, f, row.names = FALSE)
constraints_science <- loadConstraints(f,
  itempool_science, itemattrib_science)
file.remove(f)
```

---

loadItemPool *Load item pool*

Description

loadItemPool is a data loading function for creating an item_pool object. loadItemPool can read item parameters and standard errors from a data.frame or a .csv file.

Usage

```
loadItemPool(ipar, ipar_se = NULL, unique = FALSE)
```

Arguments

ipar	item parameters. Can be a data.frame or the file path of a .csv file. The content should at least include columns 'ID' and 'MODEL'.
ipar_se	(optional) standard errors. Can be a data.frame or the file path of a .csv file.
unique	if TRUE, item IDs must be unique to create a valid item_pool object. (default = FALSE)

Value

loadItemPool returns an item_pool object.

- ni the number of items in the pool.
- max_cat the maximum number of response categories across all items in the pool.
- index the numeric item index of each item.
- id the item ID string of each item.
- model the object class names of each item representing an item model type. Can be item_1PL, item_2PL, item_3PL, item_PC, item_GPC, or item_GR.
- NCAT the number of response categories of each item.
- parms a list containing the item object of each item.
- ipar a matrix containing all item parameters.
- se a matrix containing all item parameter standard errors. The values will be 0 if the argument ipar_se was not supplied.
- raw the original input ipar argument used to create this object.
- raw_se the original input ipar_se argument used to create this object. If the argument was not supplied, this will be in the same structure with the ipar argument but the item parameter values will be filled with 0s.
- unique the original input unique argument used to create this object.

See Also

dataset_science, dataset_reading, dataset_fatigue, dataset_bayes for examples.

Examples

```
## Read from data.frame:
itempool_science <- loadItemPool(itempool_science_data)

## Read from file: write to tempdir() for illustration and clean afterwards
f <- file.path(tempdir(), "itempool_science.csv")
write.csv(itempool_science_data, f, row.names = FALSE)
itempool_science <- loadItemPool(f)
file.remove(f)
```

---

logitHyperPars *Convert mean and standard deviation into logit-normal distribution parameters*

Description

logitHyperPars is a function for calculating parameters for a logit-normal distribution, such that the distribution yields desired mean and standard deviation. Used for sampling the c-parameter.

Usage

```
logitHyperPars(mean, sd)
```

Arguments

mean	the desired mean.
sd	the desired standard deviation.

Value

logitHyperPars returns two values. These can be directly supplied to rlogitnorm.

Examples

```
pars <- logitHyperPars(0.2, 0.1)
x <- logitnorm::rlogitnorm(1000000, pars[1], pars[2])
mean(x) # close to 0.2
sd(x)   # close to 0.1
```

---

makeConstraintsByEachPartition

make constraints objects from Split() solution indices

Description

makeConstraintsByEachPartition is a helper function for making constraints objects from Split solution indices.

Usage

```
makeConstraintsByEachPartition(constraints, solution_per_bin)
```

Arguments

constraints	a constraints object representing test specifications. Use loadConstraints for this.
solution_per_bin	a list containing item/stimulus indices for each partition. This accepts a list stored in the output slot of an output_Split object.

Value

makeConstraintsByEachPartition returns a list of constraints objects.

makeItemPoolCluster *Create an item pool cluster object*

Description

Create a item_pool_cluster object.

item_pool_cluster1 == item_pool_cluster2 tests equality of two item_pool_cluster objects.

Usage

```
makeItemPoolCluster(x, ..., names = NULL)

## S4 method for signature 'item_pool'
makeItemPoolCluster(x, ..., names = NULL)

## S3 method for class 'item_pool_cluster'
item_pool_cluster1 == item_pool_cluster2
```

Arguments

x, ...	item_pool objects.
names	(optional) names to use for item_pool.
item_pool_cluster1	an item_pool_cluster object.
item_pool_cluster2	an item_pool_cluster object.

Examples

```
cluster <- makeItemPoolCluster(itempool_science, itempool_reading)
cluster1 <- makeItemPoolCluster(itempool_science, itempool_reading)
cluster2 <- makeItemPoolCluster(cluster1@pools[[1]], cluster1@pools[[2]])
cluster1 == cluster2  ## TRUE
```

makeSimulationDataCache *Create a simulation data cache object*

Description

makeSimulationDataCache is a function for creating a simulation_data_cache object. This is used in Shadow to make all necessary data (e.g., item information, response data) prior to the main simulation.

Usage

```
makeSimulationDataCache(
  item_pool,
  info_type = "FISHER",
  theta_grid = seq(-4, 4, 0.1),
  seed = NULL,
  true_theta = NULL,
  response_data = NULL
)

## S4 method for signature 'item_pool'
makeSimulationDataCache(
  item_pool,
  info_type = "FISHER",
  theta_grid = seq(-4, 4, 0.1),
  seed = NULL,
  true_theta = NULL,
  response_data = NULL
)
```

Arguments

item_pool	an item_pool object.
info_type	the type of information.
theta_grid	a grid of theta values.
seed	(optional) seed to use for generating response data if needed.
true_theta	(optional) true theta values of all simulees.
response_data	(optional) response data on all items for all simulees.

---

makeTest *Create a test object*

Description

makeTest is a function for creating a test object. This is used to make all necessary data (e.g., item information, response data) prior to the main simulation. This function is only kept for backwards compatibility. The functionality of this function is superseded by makeSimulationDataCache.

Usage

```
makeTest(
  object,
  theta = seq(-4, 4, 0.1),
  info_type = "FISHER",
  true_theta = NULL
```

```
)

## S4 method for signature 'item_pool'
makeTest(
  object,
  theta = seq(-4, 4, 0.1),
  info_type = "FISHER",
  true_theta = NULL
)
```

Arguments

object	an item_pool object.
theta	a grid of theta values.
info_type	the type of information.
true_theta	(optional) true theta values to simulate response data.

Examples

```
test <- makeTest(itempool_science, seq(-3, 3, 1))
```

---

makeTestCluster *Create a test cluster object*

Description

makeTestCluster is a function for creating a test_cluster object. This is used to make all necessary data (e.g., item information, response data) prior to the main simulation. This function is only kept for backwards compatibility.

Usage

```
makeTestCluster(object, theta, true_theta)

## S4 method for signature 'item_pool_cluster,numeric,numeric'
makeTestCluster(object, theta, true_theta)

## S4 method for signature 'item_pool_cluster,numeric,list'
makeTestCluster(object, theta, true_theta)
```

Arguments

object	an item_pool_cluster object.
theta	a grid of theta values.
true_theta	an optional vector of true theta values to simulate response data.

mle *Compute maximum likelihood estimates of theta*

Description

mle is a function for computing maximum likelihood estimates of theta.

Usage

```
mle(
  object,
  select = NULL,
  resp,
  start_theta = NULL,
  max_iter = 100,
  crit = 0.001,
  truncate = FALSE,
  theta_range = c(-4, 4),
  max_change = 1,
  use_step_size = FALSE,
  step_size = 0.5,
  do_Fisher = TRUE
)

## S4 method for signature 'item_pool'
mle(
  object,
  select = NULL,
  resp,
  start_theta = NULL,
  max_iter = 50,
  crit = 0.005,
  truncate = FALSE,
  theta_range = c(-4, 4),
  max_change = 1,
  use_step_size = FALSE,
  step_size = 0.5,
  do_Fisher = TRUE
)

MLE(
  object,
  select = NULL,
  start_theta = NULL,
  max_iter = 100,
  crit = 0.001,
  theta_range = c(-4, 4),
```

```
  truncate = FALSE,
  max_change = 1,
  do_Fisher = TRUE
)

## S4 method for signature 'test'
MLE(
  object,
  select = NULL,
  start_theta = NULL,
  max_iter = 100,
  crit = 0.001,
  theta_range = c(-4, 4),
  truncate = FALSE,
  max_change = 1,
  do_Fisher = TRUE
)

## S4 method for signature 'test_cluster'
MLE(object, select = NULL, start_theta = NULL, max_iter = 100, crit = 0.001)
```

Arguments

object	an item_pool object.
select	(optional) if item indices are supplied, only the specified items are used.
resp	item response on all (or selected) items in the object argument. Can be a vector, a matrix, or a data frame. length(resp) or ncol(resp) must be equal to the number of all (or selected) items.
start_theta	(optional) initial theta values. If not supplied, EAP estimates using uniform priors are used as initial values. Uniform priors are computed using the theta_range argument below, with increments of .1.
max_iter	maximum number of iterations. (default = 100)
crit	convergence criterion to use. (default = 0.001)
truncate	set TRUE to impose a bound using theta_range on the estimate. (default = FALSE)
theta_range	a range of theta values to bound the estimate. Only effective when truncate is TRUE. (default = c(-4, 4))
max_change	upper bound to impose on the absolute change in theta between iterations. Absolute changes exceeding this value will be capped to max_change. (default = 1.0)
use_step_size	set TRUE to use step_size. (default = FALSE)
step_size	upper bound to impose on the absolute change in initial theta and estimated theta. Absolute changes exceeding this value will be capped to step_size. (default = 0.5)
do_Fisher	set TRUE to use Fisher scoring instead of Newton-Raphson method. (default = TRUE)

Value

mle returns a list containing estimated values.

- th theta value.
- se standard error.
- conv TRUE if estimation converged.
- trunc TRUE if truncation was applied on th.

Examples

```
mle(itempool_fatigue, resp = resp_fatigue_data[10, ])
mle(itempool_fatigue, select = 1:20, resp = resp_fatigue_data[10, 1:20])
```

---

mlef *Compute maximum likelihood estimates of theta using fence items*

Description

mlef is a function for computing maximum likelihood estimates of theta using fence items.

Usage

```
mlef(
  object,
  select = NULL,
  resp,
  fence_slope = 5,
  fence_difficulty = c(-5, 5),
  start_theta = NULL,
  max_iter = 100,
  crit = 0.001,
  truncate = FALSE,
  theta_range = c(-4, 4),
  max_change = 1,
  use_step_size = FALSE,
  step_size = 0.5,
  do_Fisher = TRUE
)

## S4 method for signature 'item_pool'
mlef(
  object,
  select = NULL,
  resp,
  fence_slope = 5,
  fence_difficulty = c(-5, 5),
```

```
  start_theta = NULL,
  max_iter = 50,
  crit = 0.005,
  truncate = FALSE,
  theta_range = c(-4, 4),
  max_change = 1,
  use_step_size = FALSE,
  step_size = 0.5,
  do_Fisher = TRUE
)
```

Arguments

object	an item_pool object.
select	(optional) if item indices are supplied, only the specified items are used.
resp	item response on all (or selected) items in the object argument. Can be a vector, a matrix, or a data frame. length(resp) or ncol(resp) must be equal to the number of all (or selected) items.
fence_slope	the slope parameter to use on fence items. Can be one value, or two values for the lower and the upper fence respectively. (default = 5)
fence_difficulty	the difficulty parameter to use on fence items. Must have two values for the lower and the upper fence respectively. (default = c(-5, 5))
start_theta	(optional) initial theta values. If not supplied, EAP estimates using uniform priors are used as initial values. Uniform priors are computed using the theta_range argument below, with increments of .1.
max_iter	maximum number of iterations. (default = 100)
crit	convergence criterion to use. (default = 0.001)
truncate	set TRUE to impose a bound using theta_range on the estimate. (default = FALSE)
theta_range	a range of theta values to bound the estimate. Only effective when truncate is TRUE. (default = c(-4, 4))
max_change	upper bound to impose on the absolute change in theta between iterations. Absolute changes exceeding this value will be capped to max_change. (default = 1.0)
use_step_size	set TRUE to use step_size. (default = FALSE)
step_size	upper bound to impose on the absolute change in initial theta and estimated theta. Absolute changes exceeding this value will be capped to step_size. (default = 0.5)
do_Fisher	set TRUE to use Fisher scoring instead of Newton-Raphson method. (default = TRUE)

Value

mlef returns a list containing estimated values.

- th theta value.
- se standard error.
- conv TRUE if estimation converged.
- trunc TRUE if truncation was applied on th.

References

Han, K. T. (2016). Maximum likelihood score estimation method with fences for short-length tests and computerized adaptive tests. *Applied Psychological Measurement, 40*(4), 289-301.

Examples

```
mlef(itempool_fatigue, resp = resp_fatigue_data[10, ])
mlef(itempool_fatigue, select = 1:20, resp = resp_fatigue_data[10, 1:20])
```

---

output_Shadow-class *Class 'output_Shadow': adaptive assembly solution for one simulee*

Description

output_Shadow is an S4 class for representing the adaptive assembly solution for one simulee.

Slots

simulee_id the numeric ID of the simulee.

true_theta the true theta of the simulee, if was specified.

true_theta_segment the segment number of the true theta.

final_theta_est final theta estimate.

final_se_est the standard error of final_theta_est.

administered_item_index item IDs administered at each position.

administered_item_resp item responses from the simulee at each position.

administered_item_ncat the number of categories of each administered item.

administered_stimulus_index stimulus IDs administered at each position.

shadow_test_refreshed TRUE indicates the shadowtest was refreshed for the position.

shadow_test_feasible TRUE indicates the MIP was feasible with all constraints.

solve_time elapsed time in running the solver at each position.

initial_theta_est initial theta estimate.

interim_theta_est interim theta estimates at each position.

interim_se_est the standard error of the interim estimate at each position.

theta_segment_index segment numbers of interim theta estimates.

prior prior distribution, if was specified.

`prior_par` prior parameters, if were specified.

`posterior` the posterior distribution after completing test.

`posterior_sample` posterior samples of interim theta before the estimation of final theta. `mean(posterior_sample) == interim_theta_est[test_length]` holds.

`likelihood` the likelihood distribution after completing test.

`shadow_test` the list containing the item IDs within the shadowtest used in each position.

`max_cat_pool` the maximum number of response categories the item pool had.

`ni_pool` the total number of items the item pool had.

`ns_pool` the total number of stimuli the item pool had.

`test_length_constraints` the test length constraint used in assembly.

`set_based` whether the item pool was set-based.

`item_index_by_stimulus` the list of items by each stimulus the item pool had.

`output_Shadow_all-class`

Class 'output_Shadow_all': a set of adaptive assembly solutions

---

Description

`output_Shadow_all` is an S4 class for representing a set of adaptive assembly solutions.

Details

notations

- *ni* denotes the number of items in the `item_pool` object.
- *ns* denotes the number of stimuli.
- *nj* denotes the number of participants.

Slots

`call` the function call used for obtaining this object.

`output` a length-*nj* list of `output_Shadow` objects, containing the assembly results for each participant.

`final_theta_est` a length-*nj* vector containing final theta estimates for each participant.

`final_se_est` a length-*nj* vector standard errors of the final theta estimates for each participant.

`exposure_rate` a matrix containing item-level exposure rates of all items in the pool. Also contains stimulus-level exposure rates if the assembly was set-based.

`usage_matrix` a *nj* by (*ni* + *ns*) matrix representing whether the item/stimulus was administered to each participant. Stimuli representations are appended to the right side of the matrix.

`cumulative_usage_matrix` a *nj* by (*ni* + *ns*) matrix representing the number of times the item/stimulus was administered to each participant over multiple administrations.

`true_segment_count` a length-*nj* vector containing the how many examinees are now in their segment based on the true theta. This will tend to increase. This can be reproduced with true theta values alone.

`est_segment_count` a length-*nj* vector containing the how many examinees are now in their segment based on the estimated theta. This will tend to increase. This can be reproduced with estimated theta values alone.

`eligibility_stats` exposure record for diagnostics.

`check_eligibility_stats` detailed segment-wise exposure record for diagnostics. available when `config_Shadow@exposure_control$diagnostic_stats` is `TRUE`.

`no_fading_eligibility_stats` detailed segment-wise exposure record without fading for diagnostics. available when `config_Shadow@exposure_control$diagnostic_stats` is `TRUE`.

`freq_infeasible` a table representing the number of times the assembly was initially infeasible.

`pool` the `item_pool` used in the assembly.

`config` the `config_Shadow` used in the assembly.

`constraints` the `constraints` used in the assembly.

`true_theta` the `true_theta` argument used in the assembly.

`data` the `data` argument used in the assembly.

`prior` the `prior` argument used in the assembly.

`prior_par` the `prior_par` argument used in the assembly.

`adaptivity` a list of adaptivity indices.

`simulation_constants` a list containing simulation constants parsed from input.

`output_Split-class` *Class 'output_Split': partitioning solution*

Description

`output_Split` is an S4 class for representing the partitioning solution of an item pool.

Slots

`call` the function call used for obtaining this object.

`output` a list containing item/set indices of each partition.

`feasible` for partitioning into sub-pools, `TRUE` indicates the complete assignment problem was feasible.

`solve_time` elapsed time in running the solver.

`set_based` whether the item pool is set-based.

`config` the `config_Static` used in the assembly.

`constraints` the `constraints` used in the assembly.

`partition_size_range` the partition size range for splitting into sub-pools.

`partition_type` the partition type. Can be a `test` or a `pool`.

`constraints_by_each_partition` a list of `constraints` objects that represent each partition.

output_Static-class *Class 'output_Static': fixed-form assembly solution*

Description

output_Static is an S4 class for representing a fixed-form assembly solution.

Slots

call the function call used for obtaining this object.

MIP a list containing the result from MIP solver.

selected a data.frame containing the selected items and their attributes.

obj_value the objective value of the solution.

solve_time the elapsed time in running the solver.

achieved a data.frame containing attributes of the assembled test, by each constraint.

pool the item_pool used in the assembly.

config the config_Static used in the assembly.

constraints the constraints used in the assembly.

p_item *(C++) Calculate item response probability*

Description

p_*() and array_p_*() are C++ functions for calculating item response probability.

Usage

```
p_1pl(x, b)

p_2pl(x, a, b)

p_m_2pl(x, a, d)

p_3pl(x, a, b, c)

p_m_3pl(x, a, d, c)

p_pc(x, b)

p_gpc(x, a, b)

p_m_gpc(x, a, d)
```

```
p_gr(x, a, b)

p_m_gr(x, a, d)

array_p_1pl(x, b)

array_p_2pl(x, a, b)

array_p_m_2pl(x, a, d)

array_p_3pl(x, a, b, c)

array_p_m_3pl(x, a, d, c)

array_p_pc(x, b)

array_p_gpc(x, a, b)

array_p_m_gpc(x, a, d)

array_p_gr(x, a, b)

array_p_m_gr(x, a, d)
```

Arguments

x	the theta value. The number of columns should correspond to the number of dimensions. For `array_*()` functions, the number of theta values must correspond to the number of rows.
b, d	the difficulty parameter. `b` is used for unidimensional items, and `d` is used for multidimensional items.
a	the *a*-parameter.
c	the *c*-parameter.

Details

`p_*()` functions accept a single theta value, and `array_p_*()` functions accept multiple theta values.

Supports unidimensional and multidimensional models.

- `p_1pl()`, `array_p_1pl()`: 1PL models
- `p_2pl()`, `array_p_2pl()`: 2PL models
- `p_3pl()`, `array_p_3pl()`: 3PL models
- `p_pc()`, `array_p_pc()`: PC (partial credit) models
- `p_gpc()`, `array_p_gpc()`: GPC (generalized partial credit) models

- `p_gr()`, `array_p_gr()`: GR (graded response) models
- `p_m_2pl()`, `array_p_m_2pl()`: multidimensional 2PL models
- `p_m_3pl()`, `array_p_m_3pl()`: multidimensional 3PL models
- `p_m_gpc()`, `array_p_m_gpc()`: multidimensional GPC models
- `p_m_gr()`, `array_p_m_gr()`: multidimensional GR models

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
x <- 0.5

p_1pl(x, 1)
p_2pl(x, 1, 2)
p_3pl(x, 1, 2, 0.25)
p_pc(x, c(0, 1))
p_gpc(x, 2, c(0, 1))
p_gr(x, 2, c(0, 2))

x <- matrix(seq(0.1, 0.5, 0.1)) # three theta values, unidimensional

array_p_1pl(x, 1)
array_p_2pl(x, 1, 2)
array_p_3pl(x, 1, 2, 0.25)
```

```
array_p_pc(x, c(0, 1))
array_p_gpc(x, 2, c(0, 1))
array_p_gr(x, 2, c(0, 2))
```

---

plot *Extension of plot() for objects in TestDesign package*

Description

Extension of plot() for objects in TestDesign package

Usage

```
## S4 method for signature 'item_pool'
plot(
  x,
  y,
  type = "info",
  theta = seq(-3, 3, 0.1),
  info_type = "FISHER",
  plot_sum = TRUE,
  select = NULL,
  examinee_id = 1,
  position = NULL,
  theta_range = c(-5, 5),
  ylim = NULL,
  color = "blue",
  z_ci = 1.96,
  simple = TRUE,
  theta_type = "Estimated",
  color_final = "blue",
  color_stim = "red",
  segment = NULL,
  rmse = FALSE,
  use_segment_label = TRUE,
  use_par = TRUE,
  ...
)

## S4 method for signature 'output_Static'
plot(
  x,
  y,
  type = NULL,
  theta = seq(-3, 3, 0.1),
  info_type = "FISHER",
```

```
  plot_sum = TRUE,
  select = NULL,
  examinee_id = 1,
  position = NULL,
  theta_range = c(-5, 5),
  ylim = NULL,
  color = "blue",
  z_ci = 1.96,
  simple = TRUE,
  use_par = TRUE,
  ...
)

## S4 method for signature 'constraints'
plot(
  x,
  y,
  type = "info",
  theta = seq(-3, 3, 0.1),
  info_type = "FISHER",
  plot_sum = TRUE,
  select = NULL,
  examinee_id = 1,
  position = NULL,
  theta_range = c(-5, 5),
  ylim = NULL,
  color = "blue",
  z_ci = 1.96,
  simple = TRUE,
  use_par = TRUE,
  ...
)

## S4 method for signature 'output_Shadow'
plot(
  x,
  y,
  type = "audit",
  theta = seq(-3, 3, 0.1),
  info_type = "FISHER",
  plot_sum = TRUE,
  select = NULL,
  examinee_id = 1,
  theta_range = c(-5, 5),
  ylim = NULL,
  color = "blue",
  z_ci = 1.96,
  simple = FALSE,
```

```
  theta_type = "Estimated",
  use_par = TRUE,
  ...
)

## S4 method for signature 'output_Shadow_all'
plot(
  x,
  y,
  type = "audit",
  theta = seq(-3, 3, 0.1),
  info_type = "FISHER",
  plot_sum = TRUE,
  select = NULL,
  examinee_id = 1,
  position = NULL,
  theta_range = c(-5, 5),
  ylim = NULL,
  color = "blue",
  z_ci = 1.96,
  simple = FALSE,
  theta_type = "Estimated",
  color_final = "blue",
  color_stim = "red",
  segment = NULL,
  rmse = FALSE,
  use_segment_label = TRUE,
  use_par = TRUE,
  ...
)

## S4 method for signature 'output_Split'
plot(
  x,
  y,
  type = NULL,
  theta = seq(-3, 3, 0.1),
  info_type = "FISHER",
  plot_sum = TRUE,
  select = NULL,
  examinee_id = 1,
  position = NULL,
  theta_range = c(-5, 5),
  ylim = NULL,
  color = "blue",
  z_ci = 1.96,
  simple = TRUE,
  use_par = TRUE,
```

```
  ...
)
```

Arguments

x	accepts the following signatures: • item_pool: plot information and expected scores. • constraints: plot information range based on the test length constraint. • output_Static: plot information and expected scores based on the fixed assembly solution. • output_Shadow_all: plot audit trail, shadowtest chart, exposure rates, and item overlap data from the adaptive assembly solution. • output_Shadow: plot audit trail and shadowtest chart from the adaptive assembly solution.
y	not used, exists for compatibility with plot in the base R package.
type	the type of plot. • info plots information from item_pool, output_Static, and output_Shadow_all. • score plots expected scores from item_pool and output_Static. • audit plots audit trail from output_Shadow_all and output_Shadow. • shadow plots shadowtest chart from output_Shadow_all and output_Shadow. • exposure plots exposure rates from output_Shadow_all. • overlap plots item overlap data from output_Shadow_all.
theta	the theta grid to use in plotting. (default = seq(-3, 3, .1))
info_type	the type of information. Currently accepts FISHER. (default = FISHER)
plot_sum	used in item_pool objects. • if TRUE then plot pool-level values. • if FALSE then plot item-level values, and repeat for all items in the pool. • (default = TRUE)
select	used in item_pool objects. Item indices to subset.
examinee_id	used in output_Shadow and output_Shadow_all with type = 'audit' and type = 'shadow'. The examinee numeric ID to draw the plot.
position	used in output_Shadow_all with type = 'info'. The item position to draw the plot.
theta_range	used in output_Shadow and output_Shadow_all with type = 'audit'. The theta range to plot. (default = c(-5, 5))
ylim	(optional) the y-axis plot range. Used in most plot types.
color	the color of the curve.
z_ci	used in output_Shadow and output_Shadow_all with type = 'audit'. The range to use for confidence intervals. (default = 1.96)
simple	used in output_Shadow and output_Shadow_all with type = 'shadow'. If TRUE, simplify the chart by hiding unused items.

theta_type — used in output_Shadow_all with type = 'exposure'. The type of theta to determine exposure segments. Accepts Estimated or True. (default = Estimated)

color_final — used in output_Shadow_all with type = 'exposure'. The color of item-wise exposure rates, only counting the items administered in the final theta segment as exposed.

color_stim — used in output_Shadow_all with type = 'exposure' or type = 'overlap'. The color of stimulus exposure rates or stimulus overlap data.

segment — used in output_Shadow_all with type = 'exposure'. (optional) The segment index to draw the plot. Leave empty to use all segments.

rmse — used in output_Shadow_all with type = 'exposure'. If TRUE, display the RMSE value for each segment. (default = FALSE)

use_segment_label — used in output_Shadow_all with type = 'exposure'. If TRUE, display the segment label for each segment. (default = TRUE)

use_par — if FALSE, graphical parameters are not overridden inside the function. (default = TRUE)

... — arguments to pass onto plot.

Examples

```
subitempool <- itempool_science[1:8]

## Plot item information of a pool
plot(subitempool)
plot(itempool_science, select = 1:8)

## Plot expected score of a pool
plot(subitempool, type = "score")
plot(itempool_science, type = "score", select = 1:8)

## Plot assembly results from Static()
cfg <- createStaticTestConfig()
solution <- Static(cfg, constraints_science)
plot(solution)                # defaults to the objective type
plot(solution, type = "score") # plot expected scores

## Plot attainable information range from constraints
plot(constraints_science)

## Plot assembly results from Shadow()
cfg <- createShadowTestConfig()
set.seed(1)
solution <- Shadow(cfg, constraints_science, true_theta = rnorm(1))
plot(solution, type = 'audit' , examinee_id = 1)
plot(solution, type = 'shadow', examinee_id = 1, simple = TRUE)

## plot(solution, type = 'exposure')
## plot(solution, type = 'overlap')
```

print *Extension of print() for objects in TestDesign package*

Description

Extension of print() for objects in TestDesign package

Usage

```
## S4 method for signature 'item_1PL'
print(x)

## S4 method for signature 'item_2PL'
print(x)

## S4 method for signature 'item_3PL'
print(x)

## S4 method for signature 'item_PC'
print(x)

## S4 method for signature 'item_GPC'
print(x)

## S4 method for signature 'item_GR'
print(x)

## S4 method for signature 'item_pool'
print(x)

## S4 method for signature 'item_attrib'
print(x)

## S4 method for signature 'st_attrib'
print(x)

## S4 method for signature 'summary_item_attrib'
print(x)

## S4 method for signature 'summary_st_attrib'
print(x)

## S4 method for signature 'constraints'
print(x)

## S4 method for signature 'config_Static'
print(x)
```

```
## S4 method for signature 'config_Shadow'
print(x)

## S4 method for signature 'output_Static'
print(x, index_only = TRUE)

## S4 method for signature 'output_Shadow'
print(x)

## S4 method for signature 'output_Shadow_all'
print(x)

## S4 method for signature 'exposure_rate_plot'
print(x)

## S4 method for signature 'summary_item_pool'
print(x)

## S4 method for signature 'summary_constraints'
print(x)

## S4 method for signature 'summary_output_Static'
print(x, digits = 3)

## S4 method for signature 'summary_output_Shadow_all'
print(x, digits = 3)
```

Arguments

x	an object to print.
index_only	if TRUE then only print item indices. If FALSE then print all item attributes. (default = TRUE)
digits	minimal number of *significant* digits. See print.default.

RE *Calculate Relative Errors*

Description

Calculate Relative Errors.

Usage

```
RE(RMSE_foc, RMSE_ref)
```

Arguments

RMSE_foc	A vector of RMSE values for the focal group.
RMSE_ref	A vector of RMSE values for the reference group.

RMSE *Calculate Root Mean Squared Error*

Description

Calculate Root Mean Squared Error.

Usage

```
RMSE(x, y, conditional = TRUE)
```

Arguments

x	A vector of values.
y	A vector of values.
conditional	If TRUE, calculate RMSE conditional on x.

Shadow *Run adaptive test assembly*

Description

Shadow is a test assembly function for performing adaptive test assembly based on the generalized shadow-test framework.

Usage

```
Shadow(
  config,
  constraints = NULL,
  true_theta = NULL,
  data = NULL,
  prior = NULL,
  prior_par = NULL,
  exclude = NULL,
  include_items_for_estimation = NULL,
  force_solver = FALSE,
  session = NULL,
  seed = NULL,
  cumulative_usage_matrix = NULL
```

```
)

## S4 method for signature 'config_Shadow'
Shadow(
  config,
  constraints = NULL,
  true_theta = NULL,
  data = NULL,
  prior = NULL,
  prior_par = NULL,
  exclude = NULL,
  include_items_for_estimation = NULL,
  force_solver = FALSE,
  session = NULL,
  seed = NULL,
  cumulative_usage_matrix = NULL
)
```

Arguments

config	a config_Shadow object. Use createShadowTestConfig for this.
constraints	a constraints object representing test specifications. Use loadConstraints for this.
true_theta	(optional) true theta values to use in simulation. Either true_theta or data must be supplied.
data	(optional) a matrix containing item response data to use in simulation. Either true_theta or data must be supplied.
prior	(optional) density at each config@theta_grid to use as prior. Must be a length-*nq* vector or a *nj* * *nq* matrix. This overrides prior_dist and prior_par in the config. prior and prior_par cannot be used simultaneously.
prior_par	(optional) normal distribution parameters c(mean, sd) to use as prior. Must be a length-*nq* vector or a *nj* * *nq* matrix. This overrides prior_dist and prior_par in the config. prior and prior_par cannot be used simultaneously.
exclude	(optional) a list containing item names in $i and set names in $s to exclude from selection for each participant. The length of the list must be equal to the number of participants.
include_items_for_estimation	(optional) an examinee-wise list containing: • administered_item_pool items to include in theta estimation as item_pool object. • administered_item_resp item responses to include in theta estimation.
force_solver	if TRUE, do not check whether the solver is one of recommended solvers for complex problems (set-based assembly, partitioning). (default = FALSE)
session	(optional) used to communicate with Shiny app TestDesign.
seed	(optional) used to perform data generation internally.

cumulative_usage_matrix
: (optional) a *nj* by (*ni* + *ns*) matrix containing the number of times the item/stimulus was administered previously to each participant. Stimuli representations are appended to the right side of the matrix.

Value

Shadow returns an output_Shadow_all object containing assembly results.

References

van der Linden, W. J., Reese, L. M. (1998). A model for optimal constrained adaptive testing. *Applied Psychological Measurement, 22*, 259-270.

van der Linden, W. J. (1998). Optimal assembly of psychological and educational tests. *Applied Psychological Measurement, 22*, 195-211.

van der Linden, W. J. (2000). Optimal assembly of tests with item sets. *Applied Psychological Measurement, 24*, 225-240.

van der Linden, W. J. (2005). *Linear models for optimal test design.* Springer Science & Business Media.

Examples

```
config <- createShadowTestConfig()
true_theta <- rnorm(1)
solution <- Shadow(config, constraints_science, true_theta)
solution@output
```

---

show *Extension of show() for objects in TestDesign package*

Description

Extension of show() for objects in TestDesign package

Usage

```
## S4 method for signature 'item_1PL'
show(object)

## S4 method for signature 'item_2PL'
show(object)

## S4 method for signature 'item_3PL'
show(object)

## S4 method for signature 'item_PC'
show(object)
```

```
## S4 method for signature 'item_GPC'
show(object)

## S4 method for signature 'item_GR'
show(object)

## S4 method for signature 'item_pool'
show(object)

## S4 method for signature 'item_pool_cluster'
show(object)

## S4 method for signature 'item_attrib'
show(object)

## S4 method for signature 'st_attrib'
show(object)

## S4 method for signature 'constraints'
show(object)

## S4 method for signature 'summary_item_pool'
show(object)

## S4 method for signature 'summary_item_attrib'
show(object)

## S4 method for signature 'summary_st_attrib'
show(object)

## S4 method for signature 'summary_constraints'
show(object)

## S4 method for signature 'config_Static'
show(object)

## S4 method for signature 'config_Shadow'
show(object)

## S4 method for signature 'output_Static'
show(object)

## S4 method for signature 'output_Shadow'
show(object)

## S4 method for signature 'output_Shadow_all'
show(object)
```

```
## S4 method for signature 'summary_output_Static'
show(object)

## S4 method for signature 'summary_output_Shadow_all'
show(object)

## S4 method for signature 'exposure_rate_plot'
show(object)
```

Arguments

object	an object to display.

---

simResp *Simulate item response data*

Description

simResp is a function for simulating item response data.

Usage

```
simResp(object, theta)

## S4 method for signature 'item_1PL,numeric'
simResp(object, theta)

## S4 method for signature 'item_1PL,matrix'
simResp(object, theta)

## S4 method for signature 'item_2PL,numeric'
simResp(object, theta)

## S4 method for signature 'item_2PL,matrix'
simResp(object, theta)

## S4 method for signature 'item_3PL,numeric'
simResp(object, theta)

## S4 method for signature 'item_3PL,matrix'
simResp(object, theta)

## S4 method for signature 'item_PC,numeric'
simResp(object, theta)

## S4 method for signature 'item_PC,matrix'
```

```
simResp(object, theta)

## S4 method for signature 'item_GPC,numeric'
simResp(object, theta)

## S4 method for signature 'item_GPC,matrix'
simResp(object, theta)

## S4 method for signature 'item_GR,numeric'
simResp(object, theta)

## S4 method for signature 'item_GR,matrix'
simResp(object, theta)

## S4 method for signature 'item_pool,numeric'
simResp(object, theta)

## S4 method for signature 'item_pool,matrix'
simResp(object, theta)

## S4 method for signature 'item_pool_cluster,numeric'
simResp(object, theta)

## S4 method for signature 'item_pool_cluster,list'
simResp(object, theta)
```

Arguments

object	an item or an item_pool object.
theta	theta values to use.

Details

notations • *nq* denotes the number of theta values.
• *ni* denotes the number of items in the item_pool object.

Value

item **object:** simResp returns a length *nq* vector containing simulated item response data.

item_pool **object:** simResp returns a (*nq*, *ni*) matrix containing simulated item response data.

References

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danish Institute for Educational Research.

Lord, F. M. (1952). A theory of test scores (Psychometric Monograph No. 7). Richmond, VA: Psychometric Corporation.

Birnbaum, A. (1957). *Efficient design and use of tests of mental ability for various decision-making problems* (Series Report No. 58-16. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *On the estimation of mental ability* (Series Report No. 15. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1958). *Further considerations of efficiency in tests of a mental ability* (Series Report No. 17. Project No. 7755-23). Randolph Air Force Base, TX: USAF School of Aviation Medicine.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In Lord, F. M., Novick, M. R. (eds.), *Statistical Theories of Mental Test Scores*, 397-479. Reading, MA: Addison-Wesley.

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*(2), 149-174.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*(4), 561-573.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*(2), 159-176.

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph, 17*.

Examples

```
item_1    <- new("item_1PL", difficulty = 0.5)
item_2    <- new("item_2PL", slope = 1.0, difficulty = 0.5)
item_3    <- new("item_3PL", slope = 1.0, difficulty = 0.5, guessing = 0.2)
item_4    <- new("item_PC", threshold = c(-1, 0, 1), ncat = 4)
item_5    <- new("item_GPC", slope = 1.2, threshold = c(-0.8, -1.0, 0.5), ncat = 4)
item_6    <- new("item_GR", slope = 0.9, category = c(-1, 0, 1), ncat = 4)

sim_item_1 <- simResp(item_1, seq(-3, 3, 1))
sim_item_2 <- simResp(item_2, seq(-3, 3, 1))
sim_item_3 <- simResp(item_3, seq(-3, 3, 1))
sim_item_4 <- simResp(item_4, seq(-3, 3, 1))
sim_item_5 <- simResp(item_5, seq(-3, 3, 1))
sim_item_6 <- simResp(item_6, seq(-3, 3, 1))
sim_pool   <- simResp(itempool_science, seq(-3, 3, 1))
```

---

simulation_data_cache-class

Class 'simulation_data_cache': data cache for Shadow()

Description

simulation_data_cache is an S4 class for representing data cache for Shadow().

Slots

`item_pool` the `item_pool` object.

`theta_grid` the theta grid to use as quadrature points.

`prob_grid` the list containing item response probabilities at theta quadratures.

`info_grid` the matrix containing item information values at theta quadratures.

`max_info` the maximum value of `info_grid`.

`true_theta` (optional) the true theta values.

`response_data` (optional) the matrix containing item responses.

`Split` *Split an item pool into partitions*

Description

`Split` is a function for splitting a pool into multiple parallel tests or pools. When constructing parallel tests, each test is constructed to satisfy all constraints. When constructing parallel pools, each pool is constructed so that it contains a test that satisfies all constraints.

Usage

```
Split(
  config,
  constraints,
  n_partition,
  partition_type,
  partition_size_range = NULL,
  n_maximum_partitions_per_item = 1,
  force_solver = FALSE
)

## S4 method for signature 'config_Static'
Split(
  config,
  constraints,
  n_partition,
  partition_type,
  partition_size_range = NULL,
  n_maximum_partitions_per_item = 1,
  force_solver = FALSE
)
```

Arguments

config	a config_Static object. Use createStaticTestConfig for this.
constraints	a constraints object representing test specifications. Use loadConstraints for this.
n_partition	the number of partitions to create.
partition_type	test to create tests, or pool to create pools.
partition_size_range	(optional) two integer values for the desired range for the size of a partition. Has no effect when partition_type is test. For discrete item pools, the default partition size is (pool size / number of partitions). For set-based item pools, the default partition size is (pool size / number of partitions) +/- smallest set size.
n_maximum_partitions_per_item	(optional) the number of times an item can be assigned to a partition. Setting this to 1 is equivalent to requiring all partitions to be mutually exclusive. A caveat is that when this is equal to n_partition, the assembled partitions will be identical to each other, because Split aims to minimize the test information difference between all partitions. (default = 1)
force_solver	if TRUE, do not check whether the solver is one of recommended solvers for complex problems (set-based assembly, partitioning). (default = FALSE)

Value

Split returns an output_Split object containing item/set indices of created tests/pools.

Examples

```
## Not run:
config <- createStaticTestConfig(MIP = list(solver = "RSYMPHONY"))
constraints <- constraints_science[1:10]

solution <- Split(config, constraints, n_partition = 4, partition_type = "test"))
plot(solution)
solution <- Split(config, constraints, n_partition = 4, partition_type = "pool"))
plot(solution)

## End(Not run)
```

---

st_attrib-class *Load set/stimulus/passage attributes*

Description

loadStAttrib is a data loading function for creating an st_attrib object. loadStAttrib can read itemset-level attributes from a data.frame or a .csv file.

Usage

```
loadStAttrib(object, item_attrib)
```

Arguments

object	itemset-level attributes. Can be a data.frame or the file path of a .csv file. The content should at least include an 'STID' column that matches with itemset IDs (the 'STID' column) of the item_attrib object.
item_attrib	an item_attrib object. Use loadItemAttrib for this.

Value

loadStAttrib returns a st_attrib object.

- data a data.frame containing itemset-level attributes.

See Also

dataset_reading for examples.

Examples

```
## Read from data.frame:
itempool_reading   <- loadItemPool(itempool_reading_data)
itemattrib_reading <- loadItemAttrib(itemattrib_reading_data, itempool_reading)
stimattrib_reading <- loadStAttrib(stimattrib_reading_data, itemattrib_reading)

## Read from file: write to tempdir() for illustration and clean afterwards
f <- file.path(tempdir(), "stimattrib_reading.csv")
write.csv(stimattrib_reading_data, f, row.names = FALSE)
stimattrib_reading <- loadStAttrib(f, itemattrib_reading)
file.remove(f)
```

---

st_attrib-operators *Basic functions for stimulus attribute objects*

Description

Basic functions for stimulus attribute objects

Usage

```
## S4 method for signature 'st_attrib,numeric'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'st_attrib'
dim(x)

## S4 method for signature 'st_attrib'
colnames(x)

## S4 method for signature 'st_attrib'
rownames(x)

## S4 method for signature 'st_attrib'
names(x)

## S4 method for signature 'st_attrib'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	a st_attrib object.
i, j	indices to use in subsetting.
...	not used, exists for compatibility.
drop	not used, exists for compatibility.
row.names	not used, exists for compatibility.
optional	not used, exists for compatibility.

Examples

```
x <- stimattrib_reading
x[1:10]
dim(x)
ncol(x)
nrow(x)
colnames(x)
rownames(x)
names(x)
as.data.frame(x)
```

Static *Run fixed-form test assembly*

Description

Static is a test assembly function for performing fixed-form test assembly based on the generalized shadow-test framework.

Usage

```
Static(config, constraints, force_solver = FALSE)

## S4 method for signature 'config_Static'
Static(config, constraints, force_solver = FALSE)
```

Arguments

config	a config_Static object. Use createStaticTestConfig for this.
constraints	a constraints object representing test specifications. Use loadConstraints for this.
force_solver	if TRUE, do not check whether the solver is one of recommended solvers for complex problems (set-based assembly, partitioning). (default = FALSE)

Value

Static returns a output_Static object containing the selected items.

References

van der Linden, W. J. (2005). *Linear models for optimal test design.* Springer Science & Business Media.

Examples

```
config_science <- createStaticTestConfig(
  list(
    method = "MAXINFO",
    target_location = c(-1, 1)
  )
)
solution <- Static(config_science, constraints_science)
```

---

summary *Extension of summary() for objects in TestDesign package*

Description

Extension of summary() for objects in TestDesign package

Usage

```
## S4 method for signature 'item_pool'
summary(object)

## S4 method for signature 'item_attrib'
summary(object)

## S4 method for signature 'st_attrib'
summary(object)

## S4 method for signature 'constraints'
summary(object)

## S4 method for signature 'output_Static'
summary(object, simple = FALSE)

## S4 method for signature 'output_Shadow_all'
summary(object, simple = FALSE)
```

Arguments

object	an object to summarize.
simple	if TRUE, do not print constraints. (default = FALSE)

Examples

```
summary(itempool_science)
summary(itemattrib_science)

cfg <- createStaticTestConfig()
solution <- Static(cfg, constraints_science)
summary(solution)
summary(solution, simple = TRUE)

cfg <- createShadowTestConfig()
solution <- Shadow(cfg, constraints_science, true_theta = seq(-1, 1, 1))
summary(solution)
summary(solution, simple = TRUE)
```

---

summary-classes *Summary classes*

Description

Summary classes

test-class *Class 'test': data cache for simulations*

Description

test is an S4 class for representing data cache for running simulations. Despite the name, this class does not represent a test and is not related to a test. That is, test length is not stored in this class. This class is only kept for backwards compatibility. The functionality of this class is superseded by simulation_data_cache.

Slots

pool the item_pool object.

theta the theta grid to use as quadrature points.

prob the list containing item response probabilities.

info the matrix containing item information values.

true_theta (optional) the true theta values.

data (optional) the matrix containing item responses.

test_cluster-class *Class 'test_cluster': data cache for simulations*

Description

test_cluster is an S4 class for representing data cache for running simulations. Despite the name, this class does not represent a series of tests and is not related to a series of tests. That is, test length is not stored in this class. This class is only kept for backwards compatibility.

Slots

nt the number of test objects in this cluster.

tests the list containing test objects.

names test ID strings for each test object.

test_operators *Basic operators for test objects*

Description

Create a subset of a test object.

Usage

```
subsetTest(x, i = NULL)

## S4 method for signature 'test,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

x	a test object.
i	item indices to use in subsetting.
j, drop, ...	not used, exists for compatibility.

TestDesign *Open TestDesign app*

Description

TestDesign is a caller function for opening the Shiny interface of TestDesign package.

Usage

```
TestDesign()
```

Examples

```
## Not run:
if (interactive()) {
  TestDesign()
}

## End(Not run)
```

testSolver *Test solver*

Description

Test solver

Usage

```
testSolver(solver)
```

Arguments

solver	a solver package name. Accepts `lpSolve`, `Rsymphony`, `highs`, `gurobi`, `Rglpk`.

Value

empty string `""` if solver works. A string containing error messages otherwise.

theta_EAP *(C++) Calculate a theta estimate using EAP (expected a posteriori) method*

Description

`theta_EAP()` and `theta_EAP_matrix()` are functions for calculating a theta estimate using EAP (expected a posteriori) method.

Usage

```
theta_EAP(theta_grid, item_parm, resp, ncat, model, prior, prior_parm)

theta_EAP_matrix(theta_grid, item_parm, resp, ncat, model, prior, prior_parm)
```

Arguments

theta_grid	theta quadrature points.
item_parm	a matrix containing item parameters.
resp	responses on each item. Must be a vector for `theta_EAP()`, and a matrix for `theta_EAP_matrix()`. Each row should represent an examinee.
ncat	a vector containing the number of response categories of each item.
model	a vector indicating item models of each item, using • 1: 1PL model • 2: 2PL model

- 3: 3PL model
- 4: PC model
- 5: GPC model
- 6: GR model

`prior` an integer indicating the type of prior distribution, using

- 1: normal distribution
- 2: uniform distribution

`prior_parm` a vector containing parameters for the prior distribution.

Details

`theta_EAP()` and `theta_EAP_matrix()` are designed for multiple items.

`theta_EAP()` is designed for one examinee, and `theta_EAP_matrix()` is designed for multiple examinees.

Currently supports unidimensional models.

Examples

```
# item parameters
item_parm <- matrix(c(
  1, NA,   NA,
  1,  2,   NA,
  1,  2, 0.25,
  0,  1,   NA,
  2,  0,    1,
  2,  0,    2),
  nrow = 6,
  byrow = TRUE
)

ncat  <- c(2, 2, 2, 3, 3, 3)
model <- c(1, 2, 3, 4, 5, 6)

# simulate response
item_parm <- as.data.frame(item_parm)
item_parm <- cbind(101:106, 1:6, item_parm)
pool <- loadItemPool(item_parm)
true_theta <- seq(-3, 3, 1)
resp <- simResp(pool, true_theta)

theta_grid <- matrix(seq(-3, 3, .1), , 1)

theta_EAP(theta_grid, pool@ipar, resp[1, ], ncat, model, 1, c(1, 2))
theta_EAP_matrix(theta_grid, pool@ipar, resp, ncat, model, 1, c(1, 2))
```

theta_EB *(C++) Calculate a theta estimate using EB (Empirical Bayes) method*

Description

`theta_EB_single()` and `theta_EB()` are functions for calculating a theta estimate using EB (Empirical Bayes) method.

Usage

```
theta_EB(
  nx,
  theta_init,
  theta_prop,
  item_parm,
  resp,
  ncat,
  model,
  prior,
  prior_parm
)

theta_EB_single(
  nx,
  theta_init,
  theta_prop,
  item_parm,
  resp,
  ncat,
  model,
  prior,
  prior_parm
)
```

Arguments

`nx`	the number of MCMC draws.
`theta_init`	the initial estimate to use.
`theta_prop`	the SD of the proposal distribution.
`item_parm`	a matrix containing item parameters. Each row should represent an item.
`resp`	a vector containing responses on each item.
`ncat`	a vector containing the number of response categories of each item.
`model`	a vector indicating item models of each item, using • `1`: 1PL model • `2`: 2PL model

- 3: 3PL model
- 4: PC model
- 5: GPC model
- 6: GR model

prior — an integer indicating the type of prior distribution, using

- 1: normal distribution
- 2: uniform distribution

prior_parm — a vector containing parameters for the prior distribution.

Details

theta_EB_single() is designed for one item, and theta_EB() is designed for multiple items.

Currently supports unidimensional models.

Examples

```
# item parameters
item_parm <- matrix(c(
  1, NA,   NA,
  1,  2,   NA,
  1,  2, 0.25,
  0,  1,   NA,
  2,  0,    1,
  2,  0,    2),
  nrow = 6,
  byrow = TRUE
)

ncat  <- c(2, 2, 2, 3, 3, 3)
model <- c(1, 2, 3, 4, 5, 6)
resp  <- c(0, 1, 0, 1, 0, 1)

nx <- 100
theta_init <- 0
theta_prop <- 1.0
set.seed(1)
theta_EB_single(nx, theta_init, theta_prop, item_parm[1, ], resp[1], ncat[1], model[1], 1, c(0, 1))
theta_EB(nx, theta_init, theta_prop, item_parm, resp, ncat, model, 1, c(0, 1))
```

---

theta_FB *(C++) Calculate a theta estimate using FB (Full Bayes) method*

Description

theta_FB_single() and theta_FB() are functions for calculating a theta estimate using FB (Full Bayes) method.

Usage

```
theta_FB(
  nx,
  theta_init,
  theta_prop,
  items_list,
  item_init,
  resp,
  ncat,
  model,
  prior,
  prior_parm
)

theta_FB_single(
  nx,
  theta_init,
  theta_prop,
  item_mcmc,
  item_init,
  resp,
  ncat,
  model,
  prior,
  prior_parm
)
```

Arguments

`nx`	the number of MCMC draws.
`theta_init`	the initial estimate to use.
`theta_prop`	the SD of the proposal distribution.
`item_init`	item parameter estimates. Must be a vector for `theta_FB_single()`, and a matrix for `theta_FB()`.
`resp`	a vector containing responses on each item.
`ncat`	a vector containing the number of response categories of each item.
`model`	a vector indicating item models of each item, using • `1`: 1PL model • `2`: 2PL model • `3`: 3PL model • `4`: PC model • `5`: GPC model • `6`: GR model
`prior`	an integer indicating the type of prior distribution, using • `1`: normal distribution

- 2: uniform distribution

`prior_parm` a vector containing parameters for the prior distribution.

`item_mcmc, items_list`
sampled item parameters. Must be a matrix for `theta_FB_single()`, and a list of matrices for `theta_FB()`.

Details

`theta_FB_single()` is designed for one item, and `theta_FB()` is designed for multiple items.

Currently supports unidimensional models.

`toggleConstraints` *Toggle constraints*

Description

`toggleConstraints` is a function for toggling individual constraints in a `constraints` object.

Usage

```
toggleConstraints(object, on = NULL, off = NULL)
```

Arguments

`object`	a `constraints` object from `loadConstraints`.
`on`	constraint indices to mark as active. Also accepts character IDs.
`off`	constraint indices to mark as inactive. Also accepts character IDs.

Value

`toggleConstraints` returns the updated `constraints` object.

Examples

```
constraints_science2 <- toggleConstraints(constraints_science, off = 32:36)
constraints_science3 <- toggleConstraints(constraints_science2, on = 32:36)
constraints_science4 <- toggleConstraints(constraints_science, off = "C32")
```

Index

∗ **datasets**
dataset_bayes, 39
dataset_fatigue, 39
dataset_reading, 40
dataset_science, 41
+.item_pool *(item_pool-operators)*, 59
-.item_pool *(item_pool-operators)*, 59
==.item_pool *(item_pool-operators)*, 59
==.item_pool_cluster *(makeItemPoolCluster)*, 68
[,constraints,numeric,ANY,ANY-method *(constraints-operators)*, 38
[,constraints,numeric-method *(constraints-operators)*, 38
[,item_attrib,numeric,ANY,ANY-method *(item_attrib-operators)*, 58
[,item_attrib,numeric-method *(item_attrib-operators)*, 58
[,item_pool,numeric,ANY,ANY-method *(item_pool-operators)*, 59
[,item_pool,numeric-method *(item_pool-operators)*, 59
[,st_attrib,numeric,ANY,ANY-method *(st_attrib-operators)*, 97
[,st_attrib,numeric-method *(st_attrib-operators)*, 97
[,test,ANY-method *(test_operators)*, 102
[,test,numeric,ANY,ANY-method *(test_operators)*, 102

a_to_alpha, 4
app, 5, *5*
array_dirinfo_m_2pl *(info_item)*, 51
array_dirinfo_m_3pl *(info_item)*, 51
array_dirinfo_m_gpc *(info_item)*, 51
array_dirinfo_m_gr *(info_item)*, 51
array_e_1pl *(e_item)*, 42
array_e_2pl *(e_item)*, 42
array_e_3pl *(e_item)*, 42
array_e_gpc *(e_item)*, 42
array_e_gr *(e_item)*, 42
array_e_pc *(e_item)*, 42
array_h_1pl *(h_item)*, 49
array_h_2pl *(h_item)*, 49
array_h_3pl *(h_item)*, 49
array_h_gpc *(h_item)*, 49
array_h_gr *(h_item)*, 49
array_h_pc *(h_item)*, 49
array_info_1pl *(info_item)*, 51
array_info_2pl *(info_item)*, 51
array_info_3pl *(info_item)*, 51
array_info_gpc *(info_item)*, 51
array_info_gr *(info_item)*, 51
array_info_m_2pl *(info_item)*, 51
array_info_m_3pl *(info_item)*, 51
array_info_m_gpc *(info_item)*, 51
array_info_m_gr *(info_item)*, 51
array_info_pc *(info_item)*, 51
array_j_1pl *(j_item)*, 61
array_j_2pl *(j_item)*, 61
array_j_3pl *(j_item)*, 61
array_j_gpc *(j_item)*, 61
array_j_gr *(j_item)*, 61
array_j_pc *(j_item)*, 61
array_p_1pl *(p_item)*, 78
array_p_2pl *(p_item)*, 78
array_p_3pl *(p_item)*, 78
array_p_gpc *(p_item)*, 78
array_p_gr *(p_item)*, 78
array_p_m_2pl *(p_item)*, 78
array_p_m_3pl *(p_item)*, 78
array_p_m_gpc *(p_item)*, 78
array_p_m_gr *(p_item)*, 78
array_p_pc *(p_item)*, 78
array_thisdirinfo_m_2pl *(info_item)*, 51
array_thisdirinfo_m_3pl *(info_item)*, 51
array_thisdirinfo_m_gpc *(info_item)*, 51
array_thisdirinfo_m_gr *(info_item)*, 51
as.data.frame,item_attrib-method

(item_attrib-operators), 58
as.data.frame,st_attrib-method
(st_attrib-operators), 97

buildConstraints, 5, *5, 6*

c,constraints-method
(constraints-operators), 38
c,item_pool-method
(item_pool-operators), 59
calc_info, 6
calc_info_EB, 8
calc_info_FB, 9
calc_info_matrix (calc_info), 6
calc_likelihood, 9
calc_likelihood_function
(calc_likelihood), 9
calc_log_likelihood (calc_likelihood), 9
calc_log_likelihood_function
(calc_likelihood), 9
calc_MI_FB, 11
calc_posterior, 12
calc_posterior_function, 12
calc_posterior_single, 13
calcEscore, 14, *14, 15*
calcEscore,item_1PL,matrix-method
(calcEscore), 14
calcEscore,item_1PL,numeric-method
(calcEscore), 14
calcEscore,item_2PL,matrix-method
(calcEscore), 14
calcEscore,item_2PL,numeric-method
(calcEscore), 14
calcEscore,item_3PL,matrix-method
(calcEscore), 14
calcEscore,item_3PL,numeric-method
(calcEscore), 14
calcEscore,item_GPC,matrix-method
(calcEscore), 14
calcEscore,item_GPC,numeric-method
(calcEscore), 14
calcEscore,item_GR,matrix-method
(calcEscore), 14
calcEscore,item_GR,numeric-method
(calcEscore), 14
calcEscore,item_PC,matrix-method
(calcEscore), 14
calcEscore,item_PC,numeric-method
(calcEscore), 14
calcEscore,item_pool,matrix-method
(calcEscore), 14
calcEscore,item_pool,numeric-method
(calcEscore), 14
calcEscore,item_pool_cluster,numeric-method
(calcEscore), 14
calcFisher, 16, *16, 17*
calcFisher,item_1PL,matrix-method
(calcFisher), 16
calcFisher,item_1PL,numeric-method
(calcFisher), 16
calcFisher,item_2PL,matrix-method
(calcFisher), 16
calcFisher,item_2PL,numeric-method
(calcFisher), 16
calcFisher,item_3PL,matrix-method
(calcFisher), 16
calcFisher,item_3PL,numeric-method
(calcFisher), 16
calcFisher,item_GPC,matrix-method
(calcFisher), 16
calcFisher,item_GPC,numeric-method
(calcFisher), 16
calcFisher,item_GR,matrix-method
(calcFisher), 16
calcFisher,item_GR,numeric-method
(calcFisher), 16
calcFisher,item_PC,matrix-method
(calcFisher), 16
calcFisher,item_PC,numeric-method
(calcFisher), 16
calcFisher,item_pool,matrix-method
(calcFisher), 16
calcFisher,item_pool,numeric-method
(calcFisher), 16
calcFisher,item_pool_cluster,numeric-method
(calcFisher), 16
calcHessian, 18, *18*, *20*
calcHessian,item_1PL,matrix,numeric-method
(calcHessian), 18
calcHessian,item_1PL,matrix-method
(calcHessian), 18
calcHessian,item_1PL,numeric,numeric-method
(calcHessian), 18
calcHessian,item_1PL,numeric-method
(calcHessian), 18
calcHessian,item_2PL,matrix,numeric-method
(calcHessian), 18

calcHessian,item_2PL,matrix-method
(calcHessian), 18
calcHessian,item_2PL,numeric,numeric-method
(calcHessian), 18
calcHessian,item_2PL,numeric-method
(calcHessian), 18
calcHessian,item_3PL,matrix,numeric-method
(calcHessian), 18
calcHessian,item_3PL,matrix-method
(calcHessian), 18
calcHessian,item_3PL,numeric,numeric-method
(calcHessian), 18
calcHessian,item_3PL,numeric-method
(calcHessian), 18
calcHessian,item_GPC,matrix,numeric-method
(calcHessian), 18
calcHessian,item_GPC,matrix-method
(calcHessian), 18
calcHessian,item_GPC,numeric,numeric-method
(calcHessian), 18
calcHessian,item_GPC,numeric-method
(calcHessian), 18
calcHessian,item_GR,matrix,numeric-method
(calcHessian), 18
calcHessian,item_GR,matrix-method
(calcHessian), 18
calcHessian,item_GR,numeric,numeric-method
(calcHessian), 18
calcHessian,item_GR,numeric-method
(calcHessian), 18
calcHessian,item_PC,matrix,numeric-method
(calcHessian), 18
calcHessian,item_PC,matrix-method
(calcHessian), 18
calcHessian,item_PC,numeric,numeric-method
(calcHessian), 18
calcHessian,item_PC,numeric-method
(calcHessian), 18
calcHessian,item_pool,numeric,numeric-method
(calcHessian), 18
calcHessian,item_pool,numeric-method
(calcHessian), 18
calcHessian,item_pool_cluster,numeric,list-method
(calcHessian), 18
calcHessian,item_pool_cluster,numeric-method
(calcHessian), 18
calcJacobian, 21, *21*, *22*
calcJacobian,item_1PL,matrix,numeric-method
(calcJacobian), 21
calcJacobian,item_1PL,matrix-method
(calcJacobian), 21
calcJacobian,item_1PL,numeric,numeric-method
(calcJacobian), 21
calcJacobian,item_1PL,numeric-method
(calcJacobian), 21
calcJacobian,item_2PL,matrix,numeric-method
(calcJacobian), 21
calcJacobian,item_2PL,matrix-method
(calcJacobian), 21
calcJacobian,item_2PL,numeric,numeric-method
(calcJacobian), 21
calcJacobian,item_2PL,numeric-method
(calcJacobian), 21
calcJacobian,item_3PL,matrix,numeric-method
(calcJacobian), 21
calcJacobian,item_3PL,matrix-method
(calcJacobian), 21
calcJacobian,item_3PL,numeric,numeric-method
(calcJacobian), 21
calcJacobian,item_3PL,numeric-method
(calcJacobian), 21
calcJacobian,item_GPC,matrix,numeric-method
(calcJacobian), 21
calcJacobian,item_GPC,matrix-method
(calcJacobian), 21
calcJacobian,item_GPC,numeric,numeric-method
(calcJacobian), 21
calcJacobian,item_GPC,numeric-method
(calcJacobian), 21
calcJacobian,item_GR,matrix,numeric-method
(calcJacobian), 21
calcJacobian,item_GR,matrix-method
(calcJacobian), 21
calcJacobian,item_GR,numeric,numeric-method
(calcJacobian), 21
calcJacobian,item_GR,numeric-method
(calcJacobian), 21
calcJacobian,item_PC,matrix,numeric-method
(calcJacobian), 21
calcJacobian,item_PC,matrix-method
(calcJacobian), 21
calcJacobian,item_PC,numeric,numeric-method
(calcJacobian), 21
calcJacobian,item_PC,numeric-method
(calcJacobian), 21
calcJacobian,item_pool,numeric,numeric-method

(calcJacobian), 21
calcJacobian,item_pool,numeric-method (calcJacobian), 21
calcJacobian,item_pool_cluster,numeric,list-method (calcJacobian), 21
calcJacobian,item_pool_cluster,numeric-method (calcJacobian), 21
calcLocation, *23, 24*
calcLocation (calcLocation-methods), 23
calcLocation,item_1PL-method (calcLocation-methods), 23
calcLocation,item_2PL-method (calcLocation-methods), 23
calcLocation,item_3PL-method (calcLocation-methods), 23
calcLocation,item_GPC-method (calcLocation-methods), 23
calcLocation,item_GR-method (calcLocation-methods), 23
calcLocation,item_PC-method (calcLocation-methods), 23
calcLocation,item_pool-method (calcLocation-methods), 23
calcLocation-methods, 23
calcLogLikelihood, 25, *25, 26*
calcLogLikelihood,item_pool,matrix,matrix-method (calcLogLikelihood), 25
calcLogLikelihood,item_pool,matrix,numeric-method (calcLogLikelihood), 25
calcLogLikelihood,item_pool,numeric,matrix-method (calcLogLikelihood), 25
calcLogLikelihood,item_pool,numeric,numeric-method (calcLogLikelihood), 25
calcProb, *17*, *24*, *26*, *28*
calcProb (calcProb-methods), 26
calcProb,item_1PL,matrix-method (calcProb-methods), 26
calcProb,item_1PL,numeric-method (calcProb-methods), 26
calcProb,item_2PL,matrix-method (calcProb-methods), 26
calcProb,item_2PL,numeric-method (calcProb-methods), 26
calcProb,item_3PL,matrix-method (calcProb-methods), 26
calcProb,item_3PL,numeric-method (calcProb-methods), 26
calcProb,item_GPC,matrix-method (calcProb-methods), 26
calcProb,item_GPC,numeric-method (calcProb-methods), 26
calcProb,item_GR,matrix-method (calcProb-methods), 26
calcProb,item_GR,numeric-method (calcProb-methods), 26
calcProb,item_PC,matrix-method (calcProb-methods), 26
calcProb,item_PC,numeric-method (calcProb-methods), 26
calcProb,item_pool,matrix-method (calcProb-methods), 26
calcProb,item_pool,numeric-method (calcProb-methods), 26
calcProb,item_pool_cluster,numeric-method (calcProb-methods), 26
calcProb-methods, 26
calculateAdaptivityMeasures, 29, *29*
checkConstraints, 30
colnames,item_attrib-method (item_attrib-operators), 58
colnames,st_attrib-method (st_attrib-operators), 97
combineConstraints (constraints-operators), 38
combineItemPool (item_pool-operators), 59
config_Shadow, *30*, *77*, *89*
config_Shadow-class, 30
config_Static, *29*, *34, 35*, *77, 78*, *96*, *99*
config_Static-class, 34
constraint, *36, 37*
constraint-class, 36
constraints, *5, 6*, *30*, *37–41*, *46*, *48*, *64, 65*, *67*, *77, 78*, *84*, *89*, *96*, *99*, *108*
constraints-class, 37
constraints-operators, 38
constraints_bayes (dataset_bayes), 39
constraints_bayes_data (dataset_bayes), 39
constraints_fatigue (dataset_fatigue), 39
constraints_fatigue_data (dataset_fatigue), 39
constraints_reading (dataset_reading), 40
constraints_reading_data

(dataset_reading), 40
constraints_science (dataset_science), 41
constraints_science_data (dataset_science), 41
createShadowTestConfig, *30*, *89*
createShadowTestConfig (config_Shadow-class), 30
createStaticTestConfig, *29*, *34*, *35*, *96*, *99*
createStaticTestConfig (config_Static-class), 34

data.frame, *37*, *39–41*, *48*, *57*, *59*, *64*, *65*, *78*, *96*, *97*
dataset_bayes, 39, *57*, *65*, *66*
dataset_fatigue, 39, *57*, *65*, *66*
dataset_reading, 40, *57*, *65*, *66*, *97*
dataset_science, 41, *57*, *65*, *66*
detectBestSolver, 41
dim,item_attrib-method (item_attrib-operators), 58
dim,st_attrib-method (st_attrib-operators), 97
dirinfo_m_2pl (info_item), 51
dirinfo_m_3pl (info_item), 51
dirinfo_m_gpc (info_item), 51
dirinfo_m_gr (info_item), 51

e_1pl (e_item), 42
e_2pl (e_item), 42
e_3pl (e_item), 42
e_gpc (e_item), 42
e_gr (e_item), 42
e_item, 42
e_m_2pl (e_item), 42
e_m_3pl (e_item), 42
e_m_gpc (e_item), 42
e_m_gr (e_item), 42
e_pc (e_item), 42
EAP (eap), 44
eap, 44, *44*, *45*
eap,item_pool-method (eap), 44
EAP,test-method (eap), 44
EAP,test_cluster-method (eap), 44

find_segment, 46

getScoreAttributes, 46, *46*
getSolution, 47
getSolution,list-method (getSolution), 47
getSolution,output_Static-method (getSolution), 47
getSolutionAttributes, 48, *48*

h_1pl (h_item), 49
h_2pl (h_item), 49
h_3pl (h_item), 49
h_gpc (h_item), 49
h_gr (h_item), 49
h_item, 49
h_m_2pl (h_item), 49
h_m_3pl (h_item), 49
h_m_gpc (h_item), 49
h_m_gr (h_item), 49
h_pc (h_item), 49

info_1pl (info_item), 51
info_2pl (info_item), 51
info_3pl (info_item), 51
info_gpc (info_item), 51
info_gr (info_item), 51
info_item, 51
info_m_2pl (info_item), 51
info_m_3pl (info_item), 51
info_m_gpc (info_item), 51
info_m_gr (info_item), 51
info_pc (info_item), 51
iparPosteriorSample, 55, *55*
item, *15*, *17*, *19*, *20*, *22*, *24*, *28*, *93*
item (item-classes), 56
item-classes, 56
item_1PL, *56*, *66*
item_1PL-class (item-classes), 56
item_2PL, *56*, *66*
item_2PL-class (item-classes), 56
item_3PL, *56*, *66*
item_3PL-class (item-classes), 56
item_attrib, *6*, *37*, *39–41*, *57*, *58*, *64*, *97*
item_attrib-class, 57
item_attrib-operators, 58
item_GPC, *56*, *66*
item_GPC-class (item-classes), 56
item_GR, *56*, *66*
item_GR-class (item-classes), 56
item_PC, *56*, *66*
item_PC-class (item-classes), 56

item_pool, *6*, *15*, *17*, *19, 20*, *22*, *24, 25*, *28*, *37*, *39–41*, *45*, *55*, *57*, *59–61*, *64–66*, *68–70*, *72*, *74*, *76–78*, *84*, *89*, *93*, *95*, *101*
item_pool-class, 59
item_pool-operators, 59
item_pool_cluster, *61*, *68*, *70*
item_pool_cluster-class, 61
itemattrib_bayes *(*dataset_bayes*)*, 39
itemattrib_bayes_data *(*dataset_bayes*)*, 39
itemattrib_fatigue *(*dataset_fatigue*)*, 39
itemattrib_fatigue_data *(*dataset_fatigue*)*, 39
itemattrib_reading *(*dataset_reading*)*, 40
itemattrib_reading_data *(*dataset_reading*)*, 40
itemattrib_science *(*dataset_science*)*, 41
itemattrib_science_data *(*dataset_science*)*, 41
itempool_bayes *(*dataset_bayes*)*, 39
itempool_bayes_data *(*dataset_bayes*)*, 39
itempool_fatigue *(*dataset_fatigue*)*, 39
itempool_fatigue_data *(*dataset_fatigue*)*, 39
itempool_reading *(*dataset_reading*)*, 40
itempool_reading_data *(*dataset_reading*)*, 40
itempool_science *(*dataset_science*)*, 41
itempool_science_data *(*dataset_science*)*, 41
itempool_se_bayes_data *(*dataset_bayes*)*, 39
itemtext_fatigue_data *(*dataset_fatigue*)*, 39

j_1pl *(*j_item*)*, 61
j_2pl *(*j_item*)*, 61
j_3pl *(*j_item*)*, 61
j_gpc *(*j_item*)*, 61
j_gr *(*j_item*)*, 61
j_item, 61
j_m_2pl *(*j_item*)*, 61
j_m_3pl *(*j_item*)*, 61
j_m_gpc *(*j_item*)*, 61
j_m_gr *(*j_item*)*, 61
j_pc *(*j_item*)*, 61

list, *48*

lnHyperPars, 63, *64*
loadConstraints, *30*, 64, *64, 65*, *67*, *89*, *96*, *99*, *108*
loadItemAttrib, *57*, *64*, *97*
loadItemAttrib *(*item_attrib-class*)*, 57
loadItemPool, *57*, *59*, *64*, 65, *65, 66*
loadStAttrib, *64*, *96, 97*
loadStAttrib *(*st_attrib-class*)*, 96
logitHyperPars, 66, *66, 67*

makeConstraintsByEachPartition, 67, *67*
makeItemPoolCluster, 68
makeItemPoolCluster,item_pool-method *(*makeItemPoolCluster*)*, 68
makeSimulationDataCache, 68, *68, 69*
makeSimulationDataCache,item_pool-method *(*makeSimulationDataCache*)*, 68
makeTest, 69, *69*
makeTest,item_pool-method *(*makeTest*)*, 69
makeTestCluster, 70, *70*
makeTestCluster,item_pool_cluster,numeric,list-method *(*makeTestCluster*)*, 70
makeTestCluster,item_pool_cluster,numeric,numeric-method *(*makeTestCluster*)*, 70
MLE *(*mle*)*, 71
mle, 71, *71*, *73*
mle,item_pool-method *(*mle*)*, 71
MLE,test-method *(*mle*)*, 71
MLE,test_cluster-method *(*mle*)*, 71
mlef, 73, *73, 74*
mlef,item_pool-method *(*mlef*)*, 73

names,item_attrib-method *(*item_attrib-operators*)*, 58
names,st_attrib-method *(*st_attrib-operators*)*, 97

OAT, *5*
OAT *(*app*)*, 5
output_Shadow, *47, 48*, *75, 76*, *84*
output_Shadow-class, 75
output_Shadow_all, *29*, *76*, *84, 85*, *90*
output_Shadow_all-class, 76
output_Split, *67*, *77*, *96*
output_Split-class, 77
output_Static, *47*, *78*, *84*, *99*
output_Static-class, 78

p_1pl *(*p_item*)*, 78

p_2pl (p_item), 78
p_3pl (p_item), 78
p_gpc (p_item), 78
p_gr (p_item), 78
p_item, 78
p_m_2pl (p_item), 78
p_m_3pl (p_item), 78
p_m_gpc (p_item), 78
p_m_gr (p_item), 78
p_pc (p_item), 78
plot, 81, *84*, *85*
plot,constraints-method (plot), 81
plot,item_pool-method (plot), 81
plot,output_Shadow-method (plot), 81
plot,output_Shadow_all-method (plot), 81
plot,output_Split-method (plot), 81
plot,output_Static-method (plot), 81
print, 86
print,config_Shadow-method (print), 86
print,config_Static-method (print), 86
print,constraints-method (print), 86
print,exposure_rate_plot-method (print), 86
print,item_1PL-method (print), 86
print,item_2PL-method (print), 86
print,item_3PL-method (print), 86
print,item_attrib-method (print), 86
print,item_GPC-method (print), 86
print,item_GR-method (print), 86
print,item_PC-method (print), 86
print,item_pool-method (print), 86
print,output_Shadow-method (print), 86
print,output_Shadow_all-method (print), 86
print,output_Static-method (print), 86
print,st_attrib-method (print), 86
print,summary_constraints-method (print), 86
print,summary_item_attrib-method (print), 86
print,summary_item_pool-method (print), 86
print,summary_output_Shadow_all-method (print), 86
print,summary_output_Static-method (print), 86
print,summary_st_attrib-method (print), 86
print.default, *87*

RE, 87
resp_fatigue_data (dataset_fatigue), 39
rlnorm, *64*
rlogitnorm, *67*
RMSE, 88
rownames,item_attrib-method (item_attrib-operators), 58
rownames,st_attrib-method (st_attrib-operators), 97

Shadow, *6*, *30*, *31*, *65*, *68*, 88, *88*, *90*
Shadow,config_Shadow-method (Shadow), 88
show, 90
show,config_Shadow-method (show), 90
show,config_Static-method (show), 90
show,constraints-method (show), 90
show,exposure_rate_plot-method (show), 90
show,item_1PL-method (show), 90
show,item_2PL-method (show), 90
show,item_3PL-method (show), 90
show,item_attrib-method (show), 90
show,item_GPC-method (show), 90
show,item_GR-method (show), 90
show,item_PC-method (show), 90
show,item_pool-method (show), 90
show,item_pool_cluster-method (show), 90
show,output_Shadow-method (show), 90
show,output_Shadow_all-method (show), 90
show,output_Static-method (show), 90
show,pool_cluster-method (show), 90
show,st_attrib-method (show), 90
show,summary_constraints-method (show), 90
show,summary_item_attrib-method (show), 90
show,summary_item_pool-method (show), 90
show,summary_output_Shadow_all-method (show), 90
show,summary_output_Static-method (show), 90
show,summary_st_attrib-method (show), 90
simResp, 92, *92*, *93*
simResp,item_1PL,matrix-method (simResp), 92
simResp,item_1PL,numeric-method (simResp), 92

simResp,item_2PL,matrix-method (simResp), 92
simResp,item_2PL,numeric-method (simResp), 92
simResp,item_3PL,matrix-method (simResp), 92
simResp,item_3PL,numeric-method (simResp), 92
simResp,item_GPC,matrix-method (simResp), 92
simResp,item_GPC,numeric-method (simResp), 92
simResp,item_GR,matrix-method (simResp), 92
simResp,item_GR,numeric-method (simResp), 92
simResp,item_PC,matrix-method (simResp), 92
simResp,item_PC,numeric-method (simResp), 92
simResp,item_pool,matrix-method (simResp), 92
simResp,item_pool,numeric-method (simResp), 92
simResp,item_pool_cluster,list-method (simResp), 92
simResp,item_pool_cluster,numeric-method (simResp), 92
simulation_data_cache, *68*, *94*, *101*
simulation_data_cache-class, 94
Split, *67*, 95, *95*, *96*
Split,config_Static-method (Split), 95
st_attrib, *6*, *37*, *40*, *64*, *96–98*
st_attrib-class, 96
st_attrib-operators, 97
Static, *6*, *35*, *65*, 98, *99*
Static,config_Static-method (Static), 98
stimattrib_reading (dataset_reading), 40
stimattrib_reading_data (dataset_reading), 40
subsetConstraints (constraints-operators), 38
subsetItemPool (item_pool-operators), 59
subsetTest (test_operators), 102
summary, 99
summary,constraints-method (summary), 99
summary,item_attrib-method (summary), 99
summary,item_pool-method (summary), 99
summary,output_Shadow_all-method (summary), 99
summary,output_Static-method (summary), 99
summary,st_attrib-method (summary), 99
summary-classes, 100
summary_constraints-class (summary-classes), 100
summary_item_attrib-class (summary-classes), 100
summary_item_pool-class (summary-classes), 100
summary_output_Shadow-class (summary-classes), 100
summary_output_Shadow_all-class (summary-classes), 100
summary_output_Static-class (summary-classes), 100
summary_st_attrib-class (summary-classes), 100

test, *45*, *69*, *101*, *102*
test-class, 101
test_cluster, *45*, *70*, *101*
test_cluster-class, 101
test_operators, 102
TestDesign, *5*, *89*, 102, *102*
testSolver, 103
theta_EAP, 103
theta_EAP_matrix (theta_EAP), 103
theta_EB, 105
theta_EB_single (theta_EB), 105
theta_FB, 106
theta_FB_single (theta_FB), 106
thisdirinfo_m_2pl (info_item), 51
thisdirinfo_m_3pl (info_item), 51
thisdirinfo_m_gpc (info_item), 51
thisdirinfo_m_gr (info_item), 51
toggleConstraints, 108, *108*