

# Package ‘TestDesign’

January 30, 2023

**Type** Package

**Title** Optimal Test Design Approach to Fixed and Adaptive Test Construction

**Version** 1.5.1

**Date** 2023-1-26

**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', '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 (>= 2.10)

**biocViews**

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

**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

**RoxygenNote** 7.2.3

**Encoding** UTF-8

**LazyData** true

**VignetteBuilder** knitr

**Collate** 'RcppExports.R' 'import.R' 'extensions.R' 'item\_class.R'  
 'calc\_prob\_functions.r' 'calc\_esome\_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'  
 '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'

**NeedsCompilation** yes

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

**Repository** CRAN

**Date/Publication** 2023-01-30 15:40:02 UTC

## R topics documented:

app . . . . .	4
a_to_alpha . . . . .	5
buildConstraints . . . . .	5
calcEscore . . . . .	6
calcFisher . . . . .	9
calcHessian . . . . .	11
calcJacobian . . . . .	14
calcLocation-methods . . . . .	16
calcLogLikelihood . . . . .	18
calcProb-methods . . . . .	19
calc_info . . . . .	22
calc_info_EB . . . . .	23
calc_info_FB . . . . .	24
calc_likelihood . . . . .	24
calc_MI_FB . . . . .	27
calc_posterior . . . . .	27
calc_posterior_function . . . . .	28
calc_posterior_single . . . . .	28

checkConstraints	29
config_Shadow-class	29
config_Static-class	33
constraint-class	35
constraints-class	36
constraints-operators	37
dataset_bayes	38
dataset_fatigue	38
dataset_reading	39
dataset_science	40
detectBestSolver	40
eap	41
e_item	42
find_segment	44
getSolution	45
getSolutionAttributes	46
h_item	47
info_item	49
iparPosteriorSample	53
item-classes	53
item_attrib-class	54
item_attrib-operators	55
item_pool-class	56
item_pool-operators	57
item_pool_cluster-class	59
j_item	59
lnHyperPars	61
loadConstraints	62
loadItemPool	63
logitHyperPars	65
makeItemPoolCluster	65
makeSimulationDataCache	66
makeTest	67
makeTestCluster	68
mle	68
mlef	71
output_Shadow-class	73
output_Shadow_all-class	74
output_Split-class	75
output_Static-class	75
plot	76
print	80
p_item	82
RE	85
RMSE	85
runAssembly	86
Shadow	87
show	89

simResp . . . . .	90
simulation_data_cache-class . . . . .	93
Split . . . . .	93
Static . . . . .	95
st_attrib-class . . . . .	96
st_attrib-operators . . . . .	97
summary . . . . .	98
summary-classes . . . . .	99
test-class . . . . .	99
TestDesign . . . . .	99
testSolver . . . . .	100
test_cluster-class . . . . .	100
test_operators . . . . .	101
theta_EAP . . . . .	101
theta_EB . . . . .	103
theta_FB . . . . .	104
toggleConstraints . . . . .	106

**Index** **107**

---

app *Open TestDesign app*

---

**Description**

[app](#) and [OAT](#) are aliases of [TestDesign](#).

**Usage**

`app()`

`OAT()`

**Details**

[TestDesign](#) is a caller function to open the Shiny interface of TestDesign package.

**Examples**

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

## End(Not run)
```

---

`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))
```

---

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

---

**Description**

`buildConstraints` is a data loading function to create 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,  
  pool = NULL,  
  constraints = 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 <a href="#">item_pool</a> object, a data.frame or the file path of a .csv file.
item_attrib	item attributes. Can be an <a href="#">item_attrib</a> object, a data.frame or the file path of a .csv file.
st_attrib	(optional) stimulus attributes. Can be an <a href="#">st_attrib</a> object, a data.frame or the file path of a .csv file.
pool	(deprecated) use item_pool argument instead.
constraints	(deprecated) use object argument instead.

**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)
```

---

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	<i>Calculate Fisher information</i>
------------	-------------------------------------

---

## Description

`calcFisher` is a function to calculate 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  $(nq, I)$  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.

A vector of Fisher information values over theta ( $nq$  values) for a single item or a matrix of dimension  $(nq, ni)$  for an "item\_pool".

### 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 to calculate 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 <a href="#">item</a> or an <a href="#">item_pool</a> object.
theta	theta values to use.
resp	the response data to use. This must be a single value for an <a href="#">item</a> , or a length $ni$ vector for an <a href="#">item_pool</a> .

### 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	<i>Calculate first derivative of log-likelihood</i>
--------------	---

---

## 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 to calculate 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	<i>Calculate log-likelihood</i>
-------------------	---------------------------------

---

## Description

[calcLogLikelihood](#) is a function to calculate 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 <a href="#">item_pool</a> 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	<i>Calculate item response probabilities</i>
------------------	--

---

## Description

`calcProb` is a function to calculate 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 <a href="#">item</a> or an <a href="#">item_pool</a> 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))

```

---

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 <ul style="list-style-type: none"> <li>• 1: 1PL model</li> <li>• 2: 2PL model</li> <li>• 3: 3PL model</li> <li>• 4: PC model</li> <li>• 5: GPC model</li> <li>• 6: GR model</li> </ul>

### 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	<i>Calculate the Fisher information using full Bayesian</i>
--------------	---

---

**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	<i>(C++) For multiple items, calculate likelihoods</i>
-----------------	--

---

**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 <ul style="list-style-type: none"> <li>• 1: 1PL model</li> <li>• 2: 2PL model</li> <li>• 3: 3PL model</li> <li>• 4: PC model</li> <li>• 5: GPC model</li> <li>• 6: GR model</li> </ul>
prior	an integer indicating the type of prior distribution, using <ul style="list-style-type: none"> <li>• 1: normal distribution</li> <li>• 2: uniform distribution</li> </ul>
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).

---

checkConstraints	<i>Check the consistency of constraints and item usage</i>
------------------	--

---

**Description**

Check the consistency of constraints and item usage.

**Usage**

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

**Arguments**

constraints	A <a href="#">constraints</a> object generated by <a href="#">loadConstraints</a> .
usage_matrix	A matrix of item usage data from <a href="#">Shadow</a> .
true_theta	A vector of true theta values.

---

config_Shadow-class	<i>Create a config_Shadow object</i>
---------------------	--------------------------------------

---

**Description**

[createShadowTestConfig](#) is a config function to create a [config\\_Shadow](#) object for Shadow test 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,
  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, 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 min-max 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 shadow test.
- `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)
- 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. (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. (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 to create 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, 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 min-imax 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	<i>Class 'constraint': a single constraint</i>
------------------	--

---

## Description

`constraint` is an S4 class to represent 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 to represent 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  $n_i + n_s$ .

`n_i` the number of items to search from.

`n_s` 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

<code>x, x1, x2</code>	a <code>constraints</code> object.
<code>i, j</code>	indices to use in subsetting.
<code>...</code>	not used, exists for compatibility.
<code>drop</code>	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	<i>Bayes dataset</i>
---------------	----------------------

---

### 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	<i>Fatigue dataset</i>
-----------------	------------------------

---

### 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	<i>Reading dataset</i>
-----------------	------------------------

---

### 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	<i>Science dataset</i>
-----------------	------------------------

---

### 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	<i>Detect best solver</i>
------------------	---------------------------

---

### 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))
```

---

eap *Compute expected a posteriori estimates of theta*

---

**Description**

`eap` is a function to compute 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**

<code>object</code>	an <code>item_pool</code> object.
<code>select</code>	(optional) if item indices are supplied, only the specified items are used.
<code>resp</code>	item response on all (or selected) items in the <code>object</code> argument. Can be a vector, a matrix, or a data frame. <code>length(resp)</code> or <code>ncol(resp)</code> 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 <code>test_cluster</code> objects. If TRUE, reset the prior distribution for each <code>test</code> 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])
```

---

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 <i>a</i> -parameter.
c	the <i>c</i> -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))
```

---

find\_segment

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

---

### Description

find\_segment() is a function to classify 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 <code>-Inf</code> , <code>Inf</code> 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)
```

---

```
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 <a href="#">output_Static</a> object or an <a href="#">output_Shadow</a> object.
examinee	(optional) the examinee index to display the solution. Used when the 'object' argument is an <a href="#">output_Shadow</a> 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 <a href="#">output_Shadow</a> 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 to retrieve constraints-related attributes from a solution.

**Usage**

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

**Arguments**

<code>constraints</code>	a <code>constraints</code> object.
<code>item_idx</code>	item indices from a solution.
<code>all_values</code>	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 <i>a</i> -parameter.
c	the <i>c</i> -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 <i>a</i> -parameter.
alpha_vec	the alpha angle vector. Used for directional information in thisdirinfo_*() and array_thisdirinfo_*().
c	the <i>c</i> -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 using standard errors*

---

### Description

`iparPosteriorSample` is a function for generating item parameter samples.

### 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)
```

---

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 to create an `item_attrib` object. `loadItemAttrib` can read item attributes a `data.frame` or a `.csv` file.

## Usage

```
loadItemAttrib(object, pool, file = NULL)
```

## Arguments

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

## 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)

## TestDesign 1.1.0 - Deprecated arguments
## Not run:
loadItemAttrib(object = "iatt.csv", pool) # is equivalent to
loadItemAttrib(file = "iatt.csv", pool) # pre 1.1.0

## End(Not run)
```

---

item\_attrib-operators *Basic functions for item attribute objects*

---

## Description

Basic functions for item attribute objects

**Usage**

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

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

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

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

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

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

**Arguments**

x	an <code>item_attr</code> 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 <- itemattr_science
x[1:10]
dim(x)
ncol(x)
nrow(x)
colnames(x)
rownames(x)
names(x)
as.data.frame(x)
```

---

item_pool-class	<i>Class 'item_pool': an item pool</i>
-----------------	--

---

**Description**

`item_pool` is an S4 class to represent 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 <a href="#">item_pool</a> 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 to represent 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 <i>a</i> -parameter.
c	the <i>c</i> -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)

```

**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	<i>Load constraints</i>
-----------------	-------------------------

---

**Description**

`loadConstraints` is a data loading function to create 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")`.

**Usage**

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

**Arguments**

object	constraint specifications. Can be a <code>data.frame</code> or the file path of a .csv file. See the vignette for the expected format.
pool	an <code>item_pool</code> object. Use <code>loadItemPool</code> for this.
item_attrib	an <code>item_attrib</code> object. Use <code>loadItemAttrib</code> for this.
st_attrib	(optional) an <code>st_attrib</code> object. Use <code>loadStAttrib</code> for this.
file	(deprecated) use object argument instead.

**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)

## TestDesign 1.1.0 - Deprecated arguments
## Not run:
loadConstraints(object = "consts.csv", pool, item_attrib) # is equivalent to
loadConstraints(file = "consts.csv", pool, item_attrib) # pre 1.1.0

## End(Not run)
```

---

loadItemPool

*Load item pool*


---

**Description**

`loadItemPool` is a data loading function to create 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, file = NULL, se_file = NULL, unique = FALSE)
```

**Arguments**

<code>ipar</code>	item parameters. Can be a <code>data.frame</code> or the file path of a <code>.csv</code> file. The content should at least include columns 'ID' and 'MODEL'.
<code>ipar_se</code>	(optional) standard errors. Can be a <code>data.frame</code> or the file path of a <code>.csv</code> file.
<code>file</code>	(deprecated) use <code>ipar</code> argument instead.

se_file	(deprecated) use ipar_se argument instead.
unique	if TRUE, item IDs must be unique to create a valid <code>item_pool</code> 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 `data.frame` 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)

## TestDesign 1.1.0 - Deprecated arguments
## Not run:
loadItemPool(ipar = "ipar.csv", ipar_se = "se.csv") # is equivalent to
loadItemPool(file = "ipar.csv", se_file = "se.csv") # pre 1.1.0

## End(Not run)
```



---

logitHyperPars	<i>Convert mean and standard deviation into logit-normal distribution parameters</i>
----------------	--

---

**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
```

---

makeItemPoolCluster	<i>Create an item pool cluster object</i>
---------------------	---

---

**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 <a href="#">item_pool</a> 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	<i>Create a test object</i>
----------	-----------------------------

---

**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 <a href="#">item_pool</a> 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	<i>Create a test cluster object</i>
-----------------	-------------------------------------

---

### 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

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

---

mle	<i>Compute maximum likelihood estimates of theta</i>
-----	--

---

### Description

`mle` is a function to compute 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 <code>item_pool</code> 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. <code>length(resp)</code> or <code>ncol(resp)</code> 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 <code>theta_range</code> 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 <code>theta_range</code> on the estimate. (default = FALSE)
theta_range	a range of theta values to bound the estimate. Only effective when <code>truncate</code> is TRUE. (default = <code>c(-4, 4)</code> )
max_change	upper bound to impose on the absolute change in theta between iterations. Absolute changes exceeding this value will be capped to <code>max_change</code> . (default = 1.0)
use_step_size	set TRUE to use <code>step_size</code> . (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 <code>step_size</code> . (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 to compute 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.

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

`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.

`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.

---

output\_Split-class      *Class 'output\_Split': partitioning solution*

---

### Description

`output_Split` is an S4 class to represent the partitioning solution of an item pool.

### Slots

`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.

---

output\_Static-class      *Class 'output\_Static': fixed-form assembly solution*

---

### Description

`output_Static` is an S4 class to represent a fixed-form assembly solution.

### Slots

`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.

---

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",
  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",  
  segment = NULL,  
  rmse = FALSE,  
  use_segment_label = TRUE,  
  use_par = TRUE,  
  theta_segment = NULL,  
  ...  
)  
  
## 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:

	<ul style="list-style-type: none"> <li>• <code>item_pool</code>: plot information and expected scores.</li> <li>• <code>constraints</code>: plot information range based on the test length constraint.</li> <li>• <code>output_Static</code>: plot information and expected scores based on the fixed assembly solution.</li> <li>• <code>output_Shadow_all</code>: plot audit trail, shadow test chart, and exposure rates from the adaptive assembly solution.</li> <li>• <code>output_Shadow</code>: plot audit trail and shadow test chart from the adaptive assembly solution.</li> </ul>
<code>y</code>	not used, exists for compatibility with <code>plot</code> in the base R package.
<code>type</code>	the type of plot. <ul style="list-style-type: none"> <li>• info plots information from <code>item_pool</code>, <code>output_Static</code>, and <code>output_Shadow_all</code>.</li> <li>• score plots expected scores from <code>item_pool</code> and <code>output_Static</code>.</li> <li>• audit plots audit trail from <code>output_Shadow_all</code> and <code>output_Shadow</code>.</li> <li>• shadow plots shadow test chart from <code>output_Shadow_all</code> and <code>output_Shadow</code>.</li> <li>• exposure plots exposure rates from <code>output_Shadow_all</code>.</li> </ul>
<code>theta</code>	the theta grid to use in plotting. (default = <code>seq(-3, 3, .1)</code> )
<code>info_type</code>	the type of information. Currently accepts FISHER. (default = FISHER)
<code>plot_sum</code>	used in <code>item_pool</code> objects. <ul style="list-style-type: none"> <li>• if TRUE then plot pool-level values.</li> <li>• if FALSE then plot item-level values, and repeat for all items in the pool.</li> <li>• (default = TRUE)</li> </ul>
<code>select</code>	used in <code>item_pool</code> objects. Item indices to subset.
<code>examinee_id</code>	used in <code>output_Shadow</code> and <code>output_Shadow_all</code> with <code>type = 'audit'</code> and <code>type = 'shadow'</code> . The examinee numeric ID to draw the plot.
<code>position</code>	used in <code>output_Shadow_all</code> with <code>type = 'info'</code> . The item position to draw the plot.
<code>theta_range</code>	used in <code>output_Shadow</code> and <code>output_Shadow_all</code> with <code>type = 'audit'</code> . The theta range to plot. (default = <code>c(-5, 5)</code> )
<code>ylim</code>	(optional) the y-axis plot range. Used in most plot types.
<code>color</code>	the color of the curve.
<code>z_ci</code>	used in <code>output_Shadow</code> and <code>output_Shadow_all</code> with <code>type = 'audit'</code> . The range to use for confidence intervals. (default = 1.96)
<code>simple</code>	used in <code>output_Shadow</code> and <code>output_Shadow_all</code> with <code>type = 'shadow'</code> . If TRUE, simplify the chart by hiding unused items.
<code>theta_type</code>	used in <code>output_Shadow_all</code> with <code>type = 'exposure'</code> . The type of theta to determine exposure segments. Accepts Estimated or True. (default = Estimated)
<code>color_final</code>	used in <code>output_Shadow_all</code> with <code>type = 'exposure'</code> . The color of item-wise exposure rates, only counting the items administered in the final theta segment as exposed.
<code>segment</code>	used in <code>output_Shadow_all</code> with <code>type = 'exposure'</code> . (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`.

theta\_segment   (deprecated) use `theta_type` argument instead.

## 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')
```

---

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 <i>*significant*</i> digits. See <a href="#">print.default</a> .

---

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 <i>a</i> -parameter.
c	the <i>c</i> -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))

```

---

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.

---

runAssembly	<i>Run Test Assembly</i>
-------------	--------------------------

---

### Description

`runAssembly` is a function to perform test assembly. This function is used internally in `Static` and `Shadow`.

### Usage

```
runAssembly(config, constraints, xdata = NULL, objective = NULL)
```

### Arguments

<code>config</code>	a <code>config_Static</code> or a <code>config_Shadow</code> object containing configuration options. Use <code>createStaticTestConfig</code> and <code>createShadowTestConfig</code> for this.
<code>constraints</code>	a <code>constraints</code> object. Use <code>loadConstraints</code> for this.
<code>xdata</code>	a list containing extra constraints in MIP form, to force-include previously administered items.
<code>objective</code>	the information value for each item in the pool.

### Value

a list containing the following entries:

- `MIP` a list containing the result from MIP solver.
- `status` the MIP status value, indicating whether an optimal solution was found.
- `shadow_test` the attributes of the selected items.
- `obj_value` the objective value of the solution.
- `solve_time` the elapsed time in running the solver.

### References

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

---

Shadow

*Run adaptive test assembly*

---

## Description

[Shadow](#) is a test assembly function to perform 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  
)  
  
## 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  
)
```

## Arguments

config	a <a href="#">config_Shadow</a> object. Use <a href="#">createShadowTestConfig</a> for this.
constraints	a <a href="#">constraints</a> object representing test specifications. Use <a href="#">loadConstraints</a> 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 <code>true_theta</code> or <code>data</code> must be supplied.
prior	(optional) density at each <code>config@theta_grid</code> to use as prior. Must be a length- $nq$ vector or a $nj * nq$ matrix. This overrides <code>prior_dist</code> and <code>prior_par</code> in the config. <code>prior</code> and <code>prior_par</code> cannot be used simultaneously.
prior_par	(optional) normal distribution parameters <code>c(mean, sd)</code> to use as prior. Must be a length- $nq$ vector or a $nj * nq$ matrix. This overrides <code>prior_dist</code> and <code>prior_par</code> in the config. <code>prior</code> and <code>prior_par</code> cannot be used simultaneously.
exclude	(optional) a list containing item names in <code>\$i</code> and set names in <code>\$s</code> 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: <ul style="list-style-type: none"> <li>• <code>administered_item_pool</code> items to include in theta estimation as <code>item_pool</code> object.</li> <li>• <code>administered_item_resp</code> item responses to include in theta estimation.</li> </ul>
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 <code>TestDesign</code> .
seed	(optional) used to perform data generation internally.

### 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	<i>Simulate item response data</i>
---------	------------------------------------

---

### Description

`simResp` is a function to simulate 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 to represent 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 to split 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,
  force_solver = FALSE
)

## S4 method for signature 'config_Static'
Split(
  config,
  constraints,
  n_partition,
  partition_type,
  partition_size_range = NULL,
  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.

**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**

[partition](#) 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)
```

---

Static	<i>Run fixed-form test assembly</i>
--------	-------------------------------------

---

## Description

`Static` is a test assembly function to perform 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

<code>config</code>	a <code>config_Static</code> object. Use <code>createStaticTestConfig</code> for this.
<code>constraints</code>	a <code>constraints</code> object representing test specifications. Use <code>loadConstraints</code> for this.
<code>force_solver</code>	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)
```

---

st\_attrib-class      *Load set/stimulus/passage attributes*

---

## Description

`loadStAttrib` is a data loading function to create an `st_attrib` object. `loadStAttrib` can read stimulus attributes a `data.frame` or a .csv file.

## Usage

```
loadStAttrib(object, item_attrib, file = NULL)
```

## Arguments

<code>object</code>	set attributes. Can be a <code>data.frame</code> or the file path of a .csv file. The content should at least include the column 'STID' referring to the column 'STID' in the data slot of the <code>item_attrib</code> object.
<code>item_attrib</code>	an <code>item_attrib</code> object. Use <code>loadItemAttrib</code> for this.
<code>file</code>	(deprecated) use <code>object</code> argument instead.

## Value

`loadStAttrib` returns a `st_attrib` object.

- data a `data.frame` containing stimulus 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)

## TestDesign 1.1.0 - Deprecated arguments
## Not run:
loadStAttrib(object = "satt.csv", item_attrib) # is equivalent to
loadStAttrib(file = "satt.csv", item_attrib) # pre 1.1.0

## End(Not run)
```



## 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 <code>st_attrib</code> 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)
```

---

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	<i>Summary classes</i>
-----------------	------------------------

---

### Description

Summary classes

---

test-class	<i>Class 'test': data cache for simulations</i>
------------	---

---

### Description

`test` is an S4 class to represent 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.

---

TestDesign	<i>Open TestDesign app</i>
------------	----------------------------

---

### Description

`TestDesign` is a caller function to open the Shiny interface of TestDesign package.

### Usage

```
TestDesign()
```

**Examples**

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

## End(Not run)
```

---

testSolver	<i>Test solver</i>
------------	--------------------

---

**Description**

Test solver

**Usage**

```
testSolver(solver)
```

**Arguments**

solver            a solver package name. Accepts lpSolve, Rsymphony, gurobi, Rglpk.

**Value**

empty string "" if solver works. A string containing error messages otherwise.

---

test_cluster-class	<i>Class 'test_cluster': data cache for simulations</i>
--------------------	---

---

**Description**

`test_cluster` is an S4 class to represent 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	<i>Basic operators for test objects</i>
----------------	---

---

**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**

<code>x</code>	a <code>test</code> object.
<code>i</code>	item indices to use in subsetting.
<code>j, drop, ...</code>	not used, exists for compatibility.

---

theta_EAP	<i>(C++) Calculate a theta estimate using EAP (expected a posteriori) method</i>
-----------	--

---

**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**

<code>theta_grid</code>	theta quadrature points.
<code>item_parm</code>	a matrix containing item parameters.
<code>resp</code>	responses on each item. Must be a vector for <code>theta_EAP()</code> , and a matrix for <code>theta_EAP_matrix()</code> . Each row should represent an examinee.
<code>ncat</code>	a vector containing the number of response categories of each item.
<code>model</code>	a vector indicating item models of each item, using <ul style="list-style-type: none"> <li>1: 1PL model</li> </ul>

	<ul style="list-style-type: none"> <li>• 2: 2PL model</li> <li>• 3: 3PL model</li> <li>• 4: PC model</li> <li>• 5: GPC model</li> <li>• 6: GR model</li> </ul>
prior	<p>an integer indicating the type of prior distribution, using</p> <ul style="list-style-type: none"> <li>• 1: normal distribution</li> <li>• 2: uniform distribution</li> </ul>
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 to calculate 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 <ul style="list-style-type: none"> <li>• 1: 1PL model</li> <li>• 2: 2PL model</li> </ul>

- 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 to calculate 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**

<code>nx</code>	the number of MCMC draws.
<code>theta_init</code>	the initial estimate to use.
<code>theta_prop</code>	the SD of the proposal distribution.
<code>item_init</code>	item parameter estimates. Must be a vector for <code>theta_FB_single()</code> , and a matrix for <code>theta_FB()</code> .
<code>resp</code>	a vector containing responses on each item.
<code>ncat</code>	a vector containing the number of response categories of each item.
<code>model</code>	a vector indicating item models of each item, using <ul style="list-style-type: none"> <li>• 1: 1PL model</li> <li>• 2: 2PL model</li> <li>• 3: 3PL model</li> <li>• 4: PC model</li> <li>• 5: GPC model</li> <li>• 6: GR model</li> </ul>
<code>prior</code>	an integer indicating the type of prior distribution, using <ul style="list-style-type: none"> <li>• 1: normal distribution</li> </ul>

- 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 to toggle 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, 38
  - dataset\_fatigue, 38
  - dataset\_reading, 39
  - dataset\_science, 40
- + .item\_pool (item\_pool-operators), 57
- .item\_pool (item\_pool-operators), 57
- == .item\_pool (item\_pool-operators), 57
- == .item\_pool\_cluster
  - (makeItemPoolCluster), 65
- [, constraints, numeric, ANY, ANY-method
  - (constraints-operators), 37
- [, constraints, numeric-method
  - (constraints-operators), 37
- [, item\_attrib, numeric, ANY, ANY-method
  - (item\_attrib-operators), 55
- [, item\_attrib, numeric-method
  - (item\_attrib-operators), 55
- [, item\_pool, numeric, ANY, ANY-method
  - (item\_pool-operators), 57
- [, item\_pool, numeric-method
  - (item\_pool-operators), 57
- [, st\_attrib, numeric, ANY, ANY-method
  - (st\_attrib-operators), 97
- [, st\_attrib, numeric-method
  - (st\_attrib-operators), 97
- [, test, ANY-method (test\_operators), 101
- [, test, numeric, ANY, ANY-method
  - (test\_operators), 101
- a\_to\_alpha, 5
- app, 4, 4
- array\_dirinfo\_m\_2pl (info\_item), 49
- array\_dirinfo\_m\_3pl (info\_item), 49
- array\_dirinfo\_m\_gpc (info\_item), 49
- array\_dirinfo\_m\_gr (info\_item), 49
- 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), 47
- array\_h\_2pl (h\_item), 47
- array\_h\_3pl (h\_item), 47
- array\_h\_gpc (h\_item), 47
- array\_h\_gr (h\_item), 47
- array\_h\_pc (h\_item), 47
- array\_info\_1pl (info\_item), 49
- array\_info\_2pl (info\_item), 49
- array\_info\_3pl (info\_item), 49
- array\_info\_gpc (info\_item), 49
- array\_info\_gr (info\_item), 49
- array\_info\_m\_2pl (info\_item), 49
- array\_info\_m\_3pl (info\_item), 49
- array\_info\_m\_gpc (info\_item), 49
- array\_info\_m\_gr (info\_item), 49
- array\_info\_pc (info\_item), 49
- array\_j\_1pl (j\_item), 59
- array\_j\_2pl (j\_item), 59
- array\_j\_3pl (j\_item), 59
- array\_j\_gpc (j\_item), 59
- array\_j\_gr (j\_item), 59
- array\_j\_pc (j\_item), 59
- array\_p\_1pl (p\_item), 82
- array\_p\_2pl (p\_item), 82
- array\_p\_3pl (p\_item), 82
- array\_p\_gpc (p\_item), 82
- array\_p\_gr (p\_item), 82
- array\_p\_m\_2pl (p\_item), 82
- array\_p\_m\_3pl (p\_item), 82
- array\_p\_m\_gpc (p\_item), 82
- array\_p\_m\_gr (p\_item), 82
- array\_p\_pc (p\_item), 82
- array\_thisdirinfo\_m\_2pl (info\_item), 49
- array\_thisdirinfo\_m\_3pl (info\_item), 49
- array\_thisdirinfo\_m\_gpc (info\_item), 49
- array\_thisdirinfo\_m\_gr (info\_item), 49
- as.data.frame, item\_attrib-method

- (item\_attrib-operators), 55
- as.data.frame, st\_attrib-method
  - (st\_attrib-operators), 97
- buildConstraints, 5, 5, 6
- c, constraints-method
  - (constraints-operators), 37
- c, item\_pool-method
  - (item\_pool-operators), 57
- calc\_info, 22
- calc\_info\_EB, 23
- calc\_info\_FB, 24
- calc\_info\_matrix (calc\_info), 22
- calc\_likelihood, 24
- calc\_likelihood\_function
  - (calc\_likelihood), 24
- calc\_log\_likelihood (calc\_likelihood), 24
- calc\_log\_likelihood\_function
  - (calc\_likelihood), 24
- calc\_MI\_FB, 27
- calc\_posterior, 27
- calc\_posterior\_function, 28
- calc\_posterior\_single, 28
- calcEScore, 6, 6, 8
- calcEScore, item\_1PL, matrix-method
  - (calcEScore), 6
- calcEScore, item\_1PL, numeric-method
  - (calcEScore), 6
- calcEScore, item\_2PL, matrix-method
  - (calcEScore), 6
- calcEScore, item\_2PL, numeric-method
  - (calcEScore), 6
- calcEScore, item\_3PL, matrix-method
  - (calcEScore), 6
- calcEScore, item\_3PL, numeric-method
  - (calcEScore), 6
- calcEScore, item\_GPC, matrix-method
  - (calcEScore), 6
- calcEScore, item\_GPC, numeric-method
  - (calcEScore), 6
- calcEScore, item\_GR, matrix-method
  - (calcEScore), 6
- calcEScore, item\_GR, numeric-method
  - (calcEScore), 6
- calcEScore, item\_PC, matrix-method
  - (calcEScore), 6
- calcEScore, item\_PC, numeric-method
  - (calcEScore), 6
- calcEScore, item\_pool, matrix-method
  - (calcEScore), 6
- calcEScore, item\_pool, numeric-method
  - (calcEScore), 6
- calcEScore, item\_pool\_cluster, numeric-method
  - (calcEScore), 6
- calcFisher, 9, 9, 10
- calcFisher, item\_1PL, matrix-method
  - (calcFisher), 9
- calcFisher, item\_1PL, numeric-method
  - (calcFisher), 9
- calcFisher, item\_2PL, matrix-method
  - (calcFisher), 9
- calcFisher, item\_2PL, numeric-method
  - (calcFisher), 9
- calcFisher, item\_3PL, matrix-method
  - (calcFisher), 9
- calcFisher, item\_3PL, numeric-method
  - (calcFisher), 9
- calcFisher, item\_GPC, matrix-method
  - (calcFisher), 9
- calcFisher, item\_GPC, numeric-method
  - (calcFisher), 9
- calcFisher, item\_GR, matrix-method
  - (calcFisher), 9
- calcFisher, item\_GR, numeric-method
  - (calcFisher), 9
- calcFisher, item\_PC, matrix-method
  - (calcFisher), 9
- calcFisher, item\_PC, numeric-method
  - (calcFisher), 9
- calcFisher, item\_pool, matrix-method
  - (calcFisher), 9
- calcFisher, item\_pool, numeric-method
  - (calcFisher), 9
- calcFisher, item\_pool\_cluster, numeric-method
  - (calcFisher), 9
- calcHessian, 11, 11, 12, 13
- calcHessian, item\_1PL, matrix, numeric-method
  - (calcHessian), 11
- calcHessian, item\_1PL, matrix-method
  - (calcHessian), 11
- calcHessian, item\_1PL, numeric, numeric-method
  - (calcHessian), 11
- calcHessian, item\_1PL, numeric-method
  - (calcHessian), 11

- calcHessian,item\_2PL,matrix,numeric-method  
(calcHessian), 11
- calcHessian,item\_2PL,matrix-method  
(calcHessian), 11
- calcHessian,item\_2PL,numeric,numeric-method  
(calcHessian), 11
- calcHessian,item\_2PL,numeric-method  
(calcHessian), 11
- calcHessian,item\_3PL,matrix,numeric-method  
(calcHessian), 11
- calcHessian,item\_3PL,matrix-method  
(calcHessian), 11
- calcHessian,item\_3PL,numeric,numeric-method  
(calcHessian), 11
- calcHessian,item\_3PL,numeric-method  
(calcHessian), 11
- calcHessian,item\_GPC,matrix,numeric-method  
(calcHessian), 11
- calcHessian,item\_GPC,matrix-method  
(calcHessian), 11
- calcHessian,item\_GPC,numeric,numeric-method  
(calcHessian), 11
- calcHessian,item\_GPC,numeric-method  
(calcHessian), 11
- calcHessian,item\_GR,matrix,numeric-method  
(calcHessian), 11
- calcHessian,item\_GR,matrix-method  
(calcHessian), 11
- calcHessian,item\_GR,numeric,numeric-method  
(calcHessian), 11
- calcHessian,item\_GR,numeric-method  
(calcHessian), 11
- calcHessian,item\_PC,matrix,numeric-method  
(calcHessian), 11
- calcHessian,item\_PC,matrix-method  
(calcHessian), 11
- calcHessian,item\_PC,numeric,numeric-method  
(calcHessian), 11
- calcHessian,item\_PC,numeric-method  
(calcHessian), 11
- calcHessian,item\_pool,numeric,numeric-method  
(calcHessian), 11
- calcHessian,item\_pool,numeric-method  
(calcHessian), 11
- calcHessian,item\_pool\_cluster,numeric,list-method  
(calcHessian), 11
- calcHessian,item\_pool\_cluster,numeric-method  
(calcHessian), 11
- calcJacobian, 14, 14, 15
- calcJacobian,item\_1PL,matrix,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_1PL,matrix-method  
(calcJacobian), 14
- calcJacobian,item\_1PL,numeric,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_1PL,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_2PL,matrix,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_2PL,matrix-method  
(calcJacobian), 14
- calcJacobian,item\_2PL,numeric,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_2PL,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_3PL,matrix,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_3PL,matrix-method  
(calcJacobian), 14
- calcJacobian,item\_3PL,numeric,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_3PL,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_GPC,matrix,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_GPC,matrix-method  
(calcJacobian), 14
- calcJacobian,item\_GPC,numeric,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_GPC,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_GR,matrix,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_GR,matrix-method  
(calcJacobian), 14
- calcJacobian,item\_GR,numeric,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_GR,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_PC,matrix,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_PC,matrix-method  
(calcJacobian), 14
- calcJacobian,item\_PC,numeric,numeric-method  
(calcJacobian), 14
- calcJacobian,item\_PC,numeric-method  
(calcJacobian), 14

- (calcJacobian), 14
- calcJacobian, item\_pool, numeric, numeric-method
  - (calcJacobian), 14
- calcJacobian, item\_pool, numeric-method
  - (calcJacobian), 14
- calcJacobian, item\_pool\_cluster, numeric, list-method
  - (calcJacobian), 14
- calcJacobian, item\_pool\_cluster, numeric-method
  - (calcJacobian), 14
- calcLocation, 16, 17
- calcLocation (calcLocation-methods), 16
- calcLocation, item\_1PL-method
  - (calcLocation-methods), 16
- calcLocation, item\_2PL-method
  - (calcLocation-methods), 16
- calcLocation, item\_3PL-method
  - (calcLocation-methods), 16
- calcLocation, item\_GPC-method
  - (calcLocation-methods), 16
- calcLocation, item\_GR-method
  - (calcLocation-methods), 16
- calcLocation, item\_PC-method
  - (calcLocation-methods), 16
- calcLocation, item\_pool-method
  - (calcLocation-methods), 16
- calcLocation-methods, 16
- calcLogLikelihood, 18, 18
- calcLogLikelihood, item\_pool, matrix, matrix-method
  - (calcLogLikelihood), 18
- calcLogLikelihood, item\_pool, matrix, numeric-method
  - (calcLogLikelihood), 18
- calcLogLikelihood, item\_pool, numeric, matrix-method
  - (calcLogLikelihood), 18
- calcLogLikelihood, item\_pool, numeric, numeric-method
  - (calcLogLikelihood), 18
- calcProb, 10, 17, 19–21
- calcProb (calcProb-methods), 19
- calcProb, item\_1PL, matrix-method
  - (calcProb-methods), 19
- calcProb, item\_1PL, numeric-method
  - (calcProb-methods), 19
- calcProb, item\_2PL, matrix-method
  - (calcProb-methods), 19
- calcProb, item\_2PL, numeric-method
  - (calcProb-methods), 19
- calcProb, item\_3PL, matrix-method
  - (calcProb-methods), 19
- calcProb, item\_3PL, numeric-method
  - (calcProb-methods), 19
- calcProb, item\_GPC, matrix-method
  - (calcProb-methods), 19
- calcProb, item\_GPC, numeric-method
  - (calcProb-methods), 19
- calcProb, item\_GR, matrix-method
  - (calcProb-methods), 19
- calcProb, item\_GR, numeric-method
  - (calcProb-methods), 19
- calcProb, item\_PC, matrix-method
  - (calcProb-methods), 19
- calcProb, item\_PC, numeric-method
  - (calcProb-methods), 19
- calcProb, item\_pool, matrix-method
  - (calcProb-methods), 19
- calcProb, item\_pool, numeric-method
  - (calcProb-methods), 19
- calcProb, item\_pool\_cluster, numeric-method
  - (calcProb-methods), 19
- calcProb-methods, 19
- checkConstraints, 29
- colnames, item\_attrib-method
  - (item\_attrib-operators), 55
- colnames, st\_attrib-method
  - (st\_attrib-operators), 97
- combineConstraints
  - (constraints-operators), 37
- combineItemPool (item\_pool-operators), 57
- config\_Shadow, 29, 74, 86, 87
- config\_Shadow-class, 29
- config\_Static, 33, 34, 75, 86, 94, 95
- config\_Static-class, 33
- constraint, 35, 36
- constraint-class, 35
- constraints, 5, 6, 29, 36–40, 46, 62, 63, 74, 75, 79, 86, 87, 94, 95, 106
- constraints-class, 36
- constraints-operators, 37
- constraints\_bayes (dataset\_bayes), 38
- constraints\_bayes\_data (dataset\_bayes), 38
- constraints\_fatigue (dataset\_fatigue), 38
- constraints\_fatigue\_data (dataset\_fatigue), 38
- constraints\_reading (dataset\_reading), 39

- constraints\_reading\_data  
(dataset\_reading), 39
- constraints\_science (dataset\_science),  
40
- constraints\_science\_data  
(dataset\_science), 40
- createShadowTestConfig, 29, 86, 87
- createShadowTestConfig  
(config\_Shadow-class), 29
- createStaticTestConfig, 33, 34, 86, 94, 95
- createStaticTestConfig  
(config\_Static-class), 33
- data.frame, 36, 38–40, 46, 54, 55, 57, 62–64,  
75, 96
- dataset\_bayes, 38, 55, 63, 64
- dataset\_fatigue, 38, 55, 63, 64
- dataset\_reading, 39, 55, 63, 64, 96
- dataset\_science, 40, 55, 63, 64
- detectBestSolver, 40
- dim, item\_attrib-method  
(item\_attrib-operators), 55
- dim, st\_attrib-method  
(st\_attrib-operators), 97
- dirinfo\_m\_2pl (info\_item), 49
- dirinfo\_m\_3pl (info\_item), 49
- dirinfo\_m\_gpc (info\_item), 49
- dirinfo\_m\_gr (info\_item), 49
- 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), 41
- eap, 41, 41, 42
- eap, item\_pool-method (eap), 41
- EAP, test-method (eap), 41
- EAP, test\_cluster-method (eap), 41
- find\_segment, 44
- getSolution, 45
- getSolution, list-method (getSolution),  
45
- getSolution, output\_Static-method  
(getSolution), 45
- getSolutionAttributes, 46, 46
- h\_1pl (h\_item), 47
- h\_2pl (h\_item), 47
- h\_3pl (h\_item), 47
- h\_gpc (h\_item), 47
- h\_gr (h\_item), 47
- h\_item, 47
- h\_m\_2pl (h\_item), 47
- h\_m\_3pl (h\_item), 47
- h\_m\_gpc (h\_item), 47
- h\_m\_gr (h\_item), 47
- h\_pc (h\_item), 47
- info\_1pl (info\_item), 49
- info\_2pl (info\_item), 49
- info\_3pl (info\_item), 49
- info\_gpc (info\_item), 49
- info\_gr (info\_item), 49
- info\_item, 49
- info\_m\_2pl (info\_item), 49
- info\_m\_3pl (info\_item), 49
- info\_m\_gpc (info\_item), 49
- info\_m\_gr (info\_item), 49
- info\_pc (info\_item), 49
- iparPosteriorSample, 53, 53
- item, 8, 10, 12, 15, 17, 20, 92
- item (item-classes), 53
- item-classes, 53
- item\_1PL, 53, 64
- item\_1PL-class (item-classes), 53
- item\_2PL, 53, 64
- item\_2PL-class (item-classes), 53
- item\_3PL, 53, 64
- item\_3PL-class (item-classes), 53
- item\_attrib, 6, 36, 38–40, 54–56, 62, 96
- item\_attrib-class, 54
- item\_attrib-operators, 55
- item\_GPC, 53, 64
- item\_GPC-class (item-classes), 53
- item\_GR, 53, 64
- item\_GR-class (item-classes), 53
- item\_PC, 53, 64
- item\_PC-class (item-classes), 53

- item\_pool, [6](#), [8](#), [10](#), [12](#), [13](#), [15](#), [17](#), [18](#), [20](#), [21](#), [36](#), [38–41](#), [53](#), [55–59](#), [62–64](#), [66](#), [67](#), [70](#), [71](#), [74](#), [75](#), [79](#), [88](#), [92](#), [93](#), [99](#)
- item\_pool-class, [56](#)
- item\_pool-operators, [57](#)
- item\_pool\_cluster, [59](#), [65](#), [66](#), [68](#)
- item\_pool\_cluster-class, [59](#)
- itemattrib\_bayes (dataset\_bayes), [38](#)
- itemattrib\_bayes\_data (dataset\_bayes), [38](#)
- itemattrib\_fatigue (dataset\_fatigue), [38](#)
- itemattrib\_fatigue\_data (dataset\_fatigue), [38](#)
- itemattrib\_reading (dataset\_reading), [39](#)
- itemattrib\_reading\_data (dataset\_reading), [39](#)
- itemattrib\_science (dataset\_science), [40](#)
- itemattrib\_science\_data (dataset\_science), [40](#)
- itempool\_bayes (dataset\_bayes), [38](#)
- itempool\_bayes\_data (dataset\_bayes), [38](#)
- itempool\_fatigue (dataset\_fatigue), [38](#)
- itempool\_fatigue\_data (dataset\_fatigue), [38](#)
- itempool\_reading (dataset\_reading), [39](#)
- itempool\_reading\_data (dataset\_reading), [39](#)
- itempool\_science (dataset\_science), [40](#)
- itempool\_science\_data (dataset\_science), [40](#)
- itempool\_se\_bayes\_data (dataset\_bayes), [38](#)
- itemtext\_fatigue\_data (dataset\_fatigue), [38](#)
  
- j\_1pl (j\_item), [59](#)
- j\_2pl (j\_item), [59](#)
- j\_3pl (j\_item), [59](#)
- j\_gpc (j\_item), [59](#)
- j\_gr (j\_item), [59](#)
- j\_item, [59](#)
- j\_m\_2pl (j\_item), [59](#)
- j\_m\_3pl (j\_item), [59](#)
- j\_m\_gpc (j\_item), [59](#)
- j\_m\_gr (j\_item), [59](#)
- j\_pc (j\_item), [59](#)
  
- list, [46](#)
- lnHyperPars, [61](#), [62](#)
  
- loadConstraints, [29](#), [62](#), [62](#), [63](#), [86](#), [87](#), [94](#), [95](#), [106](#)
- loadItemAttrib, [54](#), [55](#), [62](#), [96](#)
- loadItemAttrib (item\_attrib-class), [54](#)
- loadItemPool, [55](#), [57](#), [62](#), [63](#), [63](#), [64](#)
- loadStAttrib, [62](#), [96](#)
- loadStAttrib (st\_attrib-class), [96](#)
- logitHyperPars, [65](#), [65](#)
  
- makeItemPoolCluster, [65](#)
- makeItemPoolCluster, item\_pool-method (makeItemPoolCluster), [65](#)
- makeSimulationDataCache, [66](#), [66](#), [67](#)
- makeSimulationDataCache, item\_pool-method (makeSimulationDataCache), [66](#)
- makeTest, [67](#), [67](#)
- makeTest, item\_pool-method (makeTest), [67](#)
- makeTestCluster, [68](#), [68](#)
- makeTestCluster, item\_pool\_cluster, numeric, list-method (makeTestCluster), [68](#)
- makeTestCluster, item\_pool\_cluster, numeric, numeric-method (makeTestCluster), [68](#)
- MLE (mle), [68](#)
- mle, [68](#), [68](#), [70](#)
- mle, item\_pool-method (mle), [68](#)
- MLE, test-method (mle), [68](#)
- MLE, test\_cluster-method (mle), [68](#)
- mlef, [71](#), [71](#), [72](#)
- mlef, item\_pool-method (mlef), [71](#)
  
- names, item\_attrib-method (item\_attrib-operators), [55](#)
- names, st\_attrib-method (st\_attrib-operators), [97](#)
  
- OAT, [4](#)
- OAT (app), [4](#)
- output\_Shadow, [45](#), [73](#), [74](#), [79](#)
- output\_Shadow-class, [73](#)
- output\_Shadow\_all, [74](#), [79](#), [80](#), [88](#)
- output\_Shadow\_all-class, [74](#)
- output\_Split, [75](#), [94](#)
- output\_Split-class, [75](#)
- output\_Static, [45](#), [75](#), [79](#), [95](#)
- output\_Static-class, [75](#)
  
- p\_1pl (p\_item), [82](#)
- p\_2pl (p\_item), [82](#)
- p\_3pl (p\_item), [82](#)



- p\_gpc (p\_item), 82
- p\_gr (p\_item), 82
- p\_item, 82
- p\_m\_2pl (p\_item), 82
- p\_m\_3pl (p\_item), 82
- p\_m\_gpc (p\_item), 82
- p\_m\_gr (p\_item), 82
- p\_pc (p\_item), 82
- partition, 94
- plot, 76, 79, 80
- plot, constraints-method (plot), 76
- plot, item\_pool-method (plot), 76
- plot, output\_Shadow-method (plot), 76
- plot, output\_Shadow\_all-method (plot), 76
- plot, output\_Split-method (plot), 76
- plot, output\_Static-method (plot), 76
- print, 80
- print, config\_Shadow-method (print), 80
- print, config\_Static-method (print), 80
- print, constraints-method (print), 80
- print, exposure\_rate\_plot-method (print), 80
- print, item\_1PL-method (print), 80
- print, item\_2PL-method (print), 80
- print, item\_3PL-method (print), 80
- print, item\_attrib-method (print), 80
- print, item\_GPC-method (print), 80
- print, item\_GR-method (print), 80
- print, item\_PC-method (print), 80
- print, item\_pool-method (print), 80
- print, output\_Shadow-method (print), 80
- print, output\_Shadow\_all-method (print), 80
- print, output\_Static-method (print), 80
- print, st\_attrib-method (print), 80
- print, summary\_constraints-method (print), 80
- print, summary\_item\_attrib-method (print), 80
- print, summary\_item\_pool-method (print), 80
- print, summary\_output\_Shadow\_all-method (print), 80
- print, summary\_output\_Static-method (print), 80
- print, summary\_st\_attrib-method (print), 80
- print.default, 82
- RE, 85
- resp\_fatigue\_data (dataset\_fatigue), 38
- rlnorm, 62
- rlogitnorm, 65
- RMSE, 85
- rownames, item\_attrib-method (item\_attrib-operators), 55
- rownames, st\_attrib-method (st\_attrib-operators), 97
- runAssembly, 86, 86
- Shadow, 6, 29, 31, 63, 66, 86, 87, 87, 88
- Shadow, config\_Shadow-method (Shadow), 87
- show, 89
- show, config\_Shadow-method (show), 89
- show, config\_Static-method (show), 89
- show, constraints-method (show), 89
- show, exposure\_rate\_plot-method (show), 89
- show, item\_1PL-method (show), 89
- show, item\_2PL-method (show), 89
- show, item\_3PL-method (show), 89
- show, item\_attrib-method (show), 89
- show, item\_GPC-method (show), 89
- show, item\_GR-method (show), 89
- show, item\_PC-method (show), 89
- show, item\_pool-method (show), 89
- show, item\_pool\_cluster-method (show), 89
- show, output\_Shadow-method (show), 89
- show, output\_Shadow\_all-method (show), 89
- show, output\_Static-method (show), 89
- show, pool\_cluster-method (show), 89
- show, st\_attrib-method (show), 89
- show, summary\_constraints-method (show), 89
- show, summary\_item\_attrib-method (show), 89
- show, summary\_item\_pool-method (show), 89
- show, summary\_output\_Shadow\_all-method (show), 89
- show, summary\_output\_Static-method (show), 89
- show, summary\_st\_attrib-method (show), 89
- simResp, 90, 90, 92
- simResp, item\_1PL, matrix-method (simResp), 90
- simResp, item\_1PL, numeric-method (simResp), 90

- simResp,item\_2PL,matrix-method  
(simResp), 90
- simResp,item\_2PL,numeric-method  
(simResp), 90
- simResp,item\_3PL,matrix-method  
(simResp), 90
- simResp,item\_3PL,numeric-method  
(simResp), 90
- simResp,item\_GPC,matrix-method  
(simResp), 90
- simResp,item\_GPC,numeric-method  
(simResp), 90
- simResp,item\_GR,matrix-method  
(simResp), 90
- simResp,item\_GR,numeric-method  
(simResp), 90
- simResp,item\_PC,matrix-method  
(simResp), 90
- simResp,item\_PC,numeric-method  
(simResp), 90
- simResp,item\_pool,matrix-method  
(simResp), 90
- simResp,item\_pool,numeric-method  
(simResp), 90
- simResp,item\_pool\_cluster,list-method  
(simResp), 90
- simResp,item\_pool\_cluster,numeric-method  
(simResp), 90
- simulation\_data\_cache, 66, 93, 99
- simulation\_data\_cache-class, 93
- Split, 93, 93
- Split,config\_Static-method (Split), 93
- st\_attrib, 6, 36, 39, 62, 96, 97
- st\_attrib-class, 96
- st\_attrib-operators, 97
- Static, 6, 34, 63, 86, 95, 95
- Static,config\_Static-method (Static), 95
- stimattrib\_reading (dataset\_reading), 39
- stimattrib\_reading\_data  
(dataset\_reading), 39
- subsetConstraints  
(constraints-operators), 37
- subsetItemPool (item\_pool-operators), 57
- subsetTest (test\_operators), 101
- summary, 98
- summary, constraints-method (summary), 98
- summary, item\_attrib-method (summary), 98
- summary, item\_pool-method (summary), 98
- summary,output\_Shadow\_all-method  
(summary), 98
- summary,output\_Static-method (summary),  
98
- summary,st\_attrib-method (summary), 98
- summary-classes, 99
- summary\_constraints-class  
(summary-classes), 99
- summary\_item\_attrib-class  
(summary-classes), 99
- summary\_item\_pool-class  
(summary-classes), 99
- summary\_output\_Shadow\_all-class  
(summary-classes), 99
- summary\_output\_Static-class  
(summary-classes), 99
- summary\_st\_attrib-class  
(summary-classes), 99
- test, 42, 67, 99–101
- test-class, 99
- test\_cluster, 42, 68, 100
- test\_cluster-class, 100
- test\_operators, 101
- TestDesign, 4, 88, 99, 99
- testSolver, 100
- theta\_EAP, 101
- theta\_EAP\_matrix (theta\_EAP), 101
- theta\_EB, 103
- theta\_EB\_single (theta\_EB), 103
- theta\_FB, 104
- theta\_FB\_single (theta\_FB), 104
- thisdirinfo\_m\_2pl (info\_item), 49
- thisdirinfo\_m\_3pl (info\_item), 49
- thisdirinfo\_m\_gpc (info\_item), 49
- thisdirinfo\_m\_gr (info\_item), 49
- toggleConstraints, 106, 106