Package 'xxIRT'

August 30, 2017

Type Package

Title Practical Item Response Theory and Computer-Based Testing in R

Version 2.0.3

Date 2017-9-01

Author Xiao Luo [aut, cre]

Maintainer Xiao Luo <xluo1986@gmail.com>

Description An implementation of item response theory and computer-based testing based on the 3-parameter-logistic (3PL) model, with functions to perform (i) parameter estimation, (2) automated test assembly, (3) computerized adaptive testing (CAT) simulation, and (4) multistage assembly and simulation. More documentation at <https://github.com/xluo11/xxIRT>.

License GPL (>= 3)

Depends R (>= 3.1.0)

URL https://github.com/xluo11/xxIRT

BugReports https://github.com/xluo11/xxIRT/issues

Imports dplyr, ggplot2, graphics, lpSolveAPI, magrittr, reshape2, stats, utils

RoxygenNote 5.0.1

Suggests testthat

NeedsCompilation no

Repository CRAN

Date/Publication 2017-08-30 16:12:46 UTC

R topics documented:

`ata` *Automated Test Assembly (ATA)*

Description

`ata` creates an `ata` object

`ata_obj_relative` adds relative (maximize/minimize) objectives to LP

`ata_obj_absolute` adds absolute objectives to LP

`ata_constraint` adds a constraint to LP

`ata_item_maxselect` sets the maximum selection for items

`ata_item_enemy` adds enemy item relationship to LP

`ata_item_fixedvalue` sets a fixed value range for items

`ata_solve` solves the LP

`ata_solve_lpsolve` solves the LP using LpSolve

Usage

```
ata(pool, nforms = 1, len = NULL, maxselect = NULL)

## S3 method for class 'ata'
print(x, ...)

## S3 method for class 'ata'
plot(x, ...)

ata_obj_relative(x, coef, mode = c("max", "min"), negative = FALSE,
  flatten = NULL, forms = NULL, collapse = FALSE)

ata_obj_absolute(x, coef, target, forms = NULL, collapse = FALSE)

ata_constraint(x, coef, min = NA, max = NA, level = NULL, forms = NULL,
  collapse = FALSE)

ata_item_maxselect(x, maxselect, items = NULL)

ata_item_enemy(x, items)
```

```
ata_item_fixedvalue(x, items, min = NA, max = NA, forms = NULL,
  collapse = FALSE)

ata_solve(x, as.list = TRUE, timeout = 10, mip_gap = 0.1,
  verbose = "neutral", warning = FALSE, ...)

ata_solve_lpsolve(x, ...)
```

Arguments

pool	item pool
nforms	the number of forms
len	the test length
maxselect	the maximum selection of each item
x	the ata object
...	further arguments
coef	the coefficients of the objective function or the constraint
mode	the optimization mode: 'max' for maximization and 'min' for minimization
negative	TRUE when the expected value of the objective function is negative
flatten	the flatten parameter to make the objective function flat
forms	the indices of forms where objectives are added. NULL for all forms
collapse	TRUE to collapse all forms into one objective function
target	the target of the objective function
min	the lower bound of the constraint
max	the upper bound of the constraint
level	the level value for categorical variable
items	a vector of item indices
as.list	TRUE to return results in a list; otherwise, data frame
timeout	the time limit in seconds
mip_gap	the mip gap parameter
verbose	the message parameter
warning	TRUE to output warning message when LP is not solved

Details

The ata object stores LP definitions in obj, mat, dir, rhs, types, bounds, max. When calling ata_solve, it converts these definitions to the real LP object. If solved successfully, two results are added to the object: result (a matrix of binary selection results) and items (a list or data frame of selected items).

To maximize the LP, it is to maximize y while subject to sum(x) - y >= 0 and <= F (flatten parameter). To minimize the LP, it is to minimize y while subject to sum(x) - y <= 0 and >= F. By default,

y is non-negative. When `negative=TRUE`, y is set to be negative.
When `coef` is a pool-size or form-size numeric vector, coefficients are used directly. When `coef` is a variable name, variable values are used as coefficients. When `coef` is a numeric vector unequal to pool size, information at those points are used as coefficients.

`ata_obj_absolute` is to minimize y while subject to sum(x) + y >= T and sum(x) - y <= T.

For `ata_constraint`, set `coef` to a variable name and `level` a level of that variable to add a categorical constraint. Set `coef` to a variable name and leave `level` to default value (`NULL` or `NA`) to add a quantitative constraint. Set `coef` to a constant or a vector to add a constraint directly.

In `ata_solve`, additional control parameters will be passed into solvers. When passing control parameters to the GLPK solver, use the correct parameter name (see `?glpkAPI::glpkConstants`).

Examples

```
## Not run:
library(dplyr)
## generate a 100-item pool
pool <- model_3pl()$gendata(1, 100)$items
pool$content <- sample(1:3, 100, replace=TRUE)
pool$time <- round(rlnorm(100, log(60), .2))

## ex. 1: 6 forms, 10 items, maximize b parameter
x <- ata(pool, 6, len=10, maxselect=1)
x <- ata_obj_relative(x, "b", "max")
x <- ata_solve(x)
sapply(x$items, function(x) {
  c(mean=mean(x$b), sd=sd(x$b), min=min(x$b), max=max(x$b))
}) %>% t() %>% round(., 2)

## ex. 2: 4 forms, 10 items, minimize b parameter
x <- ata(pool, 3, len=10, maxselect=1)
x <- ata_obj_relative(x, "b", "min", negative=TRUE)
x <- ata_solve(x, as.list=FALSE, timeout=5)
group_by(x$items, form) %>%
  summarise(mean=mean(b), sd=sd(b), min=min(b), max=max(b)) %>%
  round(., 2)

## ex. 3: 2 forms, 10 items, mean(b) = 0, sd(b) = 1.0, content = (3, 3, 4)
x <- ata(pool, 2, len=10, maxselect=1) %>%
     ata_obj_absolute(pool$b, 0 * 10) %>%
     ata_obj_absolute((pool$b - 0)^2, 1 * 10) %>%
     ata_constraint("content", min=3, max=3, level=1) %>%
     ata_constraint("content", min=3, max=3, level=2) %>%
     ata_constraint("content", min=4, max=4, level=3)
x <- ata_solve(x, verbose="normal", timeout=5)
sapply(x$items, function(x) {
  c(mean=mean(x$b), sd=sd(x$b))
}) %>% t() %>% round(., 2)
```

```
# ex. 4: 2 forms, 10 items, flat TIF over [-1, 1]
x <- ata(pool, 2, len=10, maxselect=1) %>%
     ata_obj_relative(seq(-1, 1, .5), "max", flatten=0.05)
x <- ata_solve(x)
plot(x)

## End(Not run)
```

---

ataHelpers *ATA Helper Functions*

Description

ATA Helper Functions

`ata_append` adds constraint data into ata

`ata_form_index` converts input forms into actual form indices in LP

`ata_obj_coef` processes input coefficients for setting objective functions

Usage

```
ata_append(x, mat, dir, rhs)

ata_form_index(x, forms, collapse)

ata_obj_coef(x, coef, compensate)
```

Arguments

`x`	the ata object
`mat`	the coefficient
`dir`	direction
`rhs`	right-hand-side value
`forms`	the forms indices
`collapse`	`TRUE` to collapse forms
`coef`	the coefficients
`compensate`	`TRUE` to combine coefficients

cat_sim *Computerized Adaptive Testing (CAT) Simulation*

Description

cat_sim simulates CAT with user-defined algorithms

cat_estimate_default is a maximum likelihood estimator of the theta parameter

cat_estimate_mle_step is a maximum likelihood estimator of the theta parameter, with a fixed incremental/decremental change for all 1s or 0s responses

cat_estimate_eap is an expected a posteriori estimator of the theta parameter

cat_estimate_hybrid is a hybrid estimator of the theta parameter: EAP for all 1s or 0s responses, and MLE otherwise

cat_stop_default is a trifold stopping rule: it is the minimum standard error rule when stop_se is set in options, the minimum information rule when stop_mi is set in options, and the 95

cat_select_default selects the item with maximum information or the item set with maximum averaged information for the current theta estimate.

cat_select_ccat implements the constrained CAT selection algorithm. The 'ccat_perc' argument defines the percentage targets, and the 'ccat_var' argument defines the constrained variable.

cat_select_shadow implements the shadow-test CAT selection algorithm. The 'shadow_constraints' argument defines the constraints.

Usage

```
cat_sim(true, pool, ...)

cat_estimate_default(len, theta, stats, admin, pool, opts)

cat_estimate_mle_step(len, theta, stats, admin, pool, opts)

cat_estimate_eap(len, theta, stats, admin, pool, opts)

cat_estimate_hybrid(len, theta, stats, admin, pool, opts)

cat_stop_default(len, theta, stats, admin, pool, opts)

cat_select_default(len, theta, stats, admin, pool, opts)

cat_select_ccat(len, theta, stats, admin, pool, opts)

cat_select_shadow(len, theta, stats, admin, pool, opts)

## S3 method for class 'cat'
print(x, ...)
```

```
## S3 method for class 'cat'
plot(x, ...)
```

Arguments

true	the true theta
pool	the item pool (data.frame)
...	additional option parameters. See details.
len	the current test length
theta	the current theta estimate
stats	a matrix of responses and statistics
admin	a data frame of administered item pool
opts	a list of options passed in cat_sim
x	a cat object

Details

... takes additional option/control parameters from users. min and max are mandatory in order to control the minimum and maximum test length.
The selection, estimation, and stopping rules takes the same set of arguments: fucntion(len, theta, stats, admin, pool which are the current test length, the current theta estimate, the matrix of statistics, the data.frame of administered items, the item pool, and the options. To override default rules, first write new rules using the same function signature and pass the new rule and its required parameters to cat_sim() options.
The returned cat object contains the remaining item pool, the administration history, the true and estimated thetas.

Value

cat_sim returns a cat object. See details.

the estimate rule should return a numeric value

the stopping rule should return a boolean with TRUE to stop the CAT

the selection rule should return a list of the selected item and the updated pool

Examples

```
## generate item pool
pool <- model_3pl()$gendata(1, 100)$items
pool$set_id <- sample(1:30, 100, replace=TRUE)
pool$content <- sample(1:3, 100, replace=TRUE)
pool$time <- round(rlnorm(100, mean=4.1, sd=.2))
## randomesque to control exposure in selection
cat_sim(1.0, pool, min=10, max=20, randomesque=5)
## use user-defined ID variable to select item sets
cat_sim(1.0, pool, min=10, max=20, selct_id="set")
## use the mle_step estimation rule
```

```
cat_sim(1.0, pool, min=10, max=20, mle_step=.5,
    estimate_rule=cat_estimate_mle_step)
## use the hybrid estimation rule
cat_sim(1.0, pool, min=10, max=20, estimate_rule=cat_estimate_hybrid)
## use the standard error stopping rule
cat_sim(1.0, pool, min=10, max=20, stop_rule=cat_stop_default, stop_se=.25)
## use the 95% confidence interval classification stopping rule
cat_sim(1.0, pool, min=10, max=20, stop_rule=cat_stop_default, stop_cut=0)
## use the constrained CAT item selection
cat_sim(1.0, pool, min=10, max=20, select_rule=cat_select_ccat,
    ccat_var='content', ccat_perc=c('1'=.2, '2'=.3, '3'=.5))
## use the constrained CAT item selection with initial randomness
cat_sim(1.0, pool, min=10, max=20, select_rule=cat_select_ccat,
    ccat_var='content', ccat_perc=c('1'=.2, '2'=.3, '3'=.5), ccat_init_rand=5)
## use the shadow-test CAT
cons <- data.frame(var='content', level=1:3, min=3, max=5)
cons <- rbind(cons, data.frame(var='time', level=NA, min=55*10, max=65*10))
cat_sim(1.0, pool, min=10, max=10, shadow_constraints=cons, select_id="set_id")
```

`estimation` *Estimation of 3PL Model*

Description

Estimation of 3PL Model

`estimate_mle` estimates parameters using joint or maximum likelihood estimation method

`estimate_bayesian` estimates parameters using Bayesian estimation method

a helper function to process input arguments

Usage

```
estimate_mle(u, t = NULL, a = NULL, b = NULL, c = NULL, iter = 20,
  conv = 0.005, method = c("jmle", "mmle"), bound_t = 3.5, bound_a = 2,
  bound_b = 3.5, bound_c = 0.25, mmle_mu = 0, mmle_sig = 1,
  scale = c("none", "theta", "b"), scale_mean = 0, scale_sd = 1,
  debug = FALSE)

estimate_bayesian(u, t = NULL, a = NULL, b = NULL, c = NULL,
  method = c("map", "eap"), iter = 20, conv = 0.005, bound_t = 3.5,
  bound_a = 2, bound_b = 3.5, bound_c = 0.25, scale = c("none", "theta",
  "b"), scale_mean = 0, scale_sd = 1, t_mu = 0, t_sig = 1, a_mu = 0,
  a_sig = 0.2, b_mu = 0, b_sig = 1, c_alpha = 5, c_beta = 46,
  report_sd = FALSE, debug = FALSE)

estimate_check_input(u, t, a, b, c)
```

Arguments

u	a matrix of response data
t	theta parameters
a	a parameters
b	b parameters
c	c parameters
iter	the number of maximum iterations
conv	the convergence criterion
method	the estimation method of item parameters
bound_t	the bound of theta parameters
bound_a	the bound of a parameters
bound_b	the bound of b parameters
bound_c	the bound of c parameters
mmle_mu	the mean of the marginal distribution in MMLE
mmle_sig	the SD of the marginal parameters in MMLE
scale	the scaling parameter
scale_mean	the mean of the scale
scale_sd	the SD of the scale
debug	TRUE to print and report debugging information
t_mu	the mean of the prior distribution of theta parameters
t_sig	the SD of the prior distribution of theta parameters
a_mu	the mean of the prior distribution of a parameters
a_sig	the SD of the prior distribution of a parameters
b_mu	the mean of the prior distribution of b parameters
b_sig	the SD of the prior distribution of b parameters
c_alpha	the alpha of the prior distribution of c parameters
c_beta	the beta of the prior distribution of c parameters
report_sd	TRUE to report the posterior variance of thetas in EAP

Details

When the t, a, b, c parameters are NULL, they are free to be estimated; otherwise, they are fixed at the provided values. When setting values for a parameter, use numeric values to fix parameters and NA to free parameters. For instance, an argument of t=c(-1, NA, 1) means to fix the 1st and 3rd theta parameters to -1 and 1 and estimate the 2nd theta parameters. The same is true for the a-, b-, and c-parameters.

The method argument in estimate_mle controls whether to use joint (jmle) or maximum (mmle) likelihood method to estimate item parameters. The scale argument controls where to set the scale: b or theta parameters.

When debug mode is on, print and report additional information regarding the convergence over the

iterations.

The method argument in estimate_bayesian controls whether to use maximum (map) or expected (eap) a posteriori to estimate theta parameters.

Value

a list of t, a, b, c parameters

Examples

```
## Not run:
library(ggplot2)
library(dplyr)
set.seed(10001)
### generate data
data <- model_3pl()$gendata(1000, 40)
### MLE
x <- estimate_mle(data$responses, debug=TRUE)
y <- rbind(data.frame(param='t', true=data$people$theta, est=x$t),
     data.frame(param='a', true=data$items$a, est=x$a),
     data.frame(param='b', true=data$items$b, est=x$b),
     data.frame(param='c', true=data$items$c, est=x$c))
group_by(y, param) %>%
summarise(corr=cor(true, est), rmse=rmse(true, est))
ggplot(y, aes(x=true, y=est, color=param)) +
  geom_point(alpha=.5) + facet_wrap(~param, scales="free") +
  xlab("True Parameters") + ylab("Estimated Parameters") +
  theme_bw()
group_by(y, param) %>% summarise(corr=cor(true, est),
    rmse=rmse(true, est), mean=mean(est), sd=sd(est))
### Bayesian
x <- estimate_bayesian(data$responses, debug=TRUE)
y <- rbind(data.frame(param='t', true=data$people$theta, est=x$t),
     data.frame(param='a', true=data$items$a, est=x$a),
     data.frame(param='b', true=data$items$b, est=x$b),
     data.frame(param='c', true=data$items$c, est=x$c))
group_by(y, param) %>% summarise(corr=cor(true, est), rmse=rmse(true, est))
ggplot(y, aes(x=true, y=est, color=param)) +
  geom_point(alpha=.5) + facet_wrap(~param, scales="free") +
  xlab("True Parameters") + ylab("Estimated Parameters") +
  theme_bw()
group_by(y, param) %>% summarise(corr=cor(true, est),
    rmse=rmse(true, est), mean=mean(est), sd=sd(est))

## End(Not run)
```

freq *Frequency*

Description

computes the frequency and percentage of given values

Usage

```
freq(x, val = NULL)
```

Arguments

x	a vector of raw data
val	a vector of valid values, NULL for all values

Examples

```
freq(sample(1:4, 150, replace=TRUE))
freq(sample(1:4, 150, replace=TRUE), 5:1)
freq(sample(1:4, 150, replace=TRUE), 1:3)
```

irt_stats *IRT Utility Functions*

Description

`irt_stats` provides a friendly interface to compute statistics

`irt_select` subsets data in an IRT model

`irt_sample` randomly samples data in an IRT model

`irt_rescale` rescales parameters in a 3PL model

Usage

```
irt_stats(x, stats = c("prob", "info", "lik", "loglik"), summary = NULL,
  fun = NULL, ...)

irt_select(x, people_index = NULL, item_index = NULL)

irt_sample(x, n.people = NULL, n.items = NULL)

irt_rescale(x, parameter = c("theta", "b"), mean = 0, sd = 1)
```

Arguments

x	an IRT model object
stats	the statistic to be computed
summary	the summarization direction
fun	the summarization function
...	other optional arguments
people_index	the indices of people to keep
item_index	the indices of items to keep
n.people	the number of people to sample
n.items	the number of items to sample
parameter	the rescaling parameter: 'theta' or 'b'
mean	the mean of the new scale
sd	the standard deviation of the new scale

Details

In irt_stats, use stats="prob" to compute probability, stats="info" to compute information, stats="lik" to compute likelihood, and stats="loglik" to compute log-likelihood. Use summary="people" to summarize results over items and summary="items" to summarize results over people.

Examples

```
# Compute probability, information, likelihood and log-likelihood
x <- model_3pl()$gendata(20, 5)
irt_stats(x, "prob")
irt_stats(x, "prob", "people", sum)
irt_stats(x, "info")
irt_stats(x, "info", "items", sum)
irt_stats(x, "lik")
irt_stats(x, "loglik")
# subset
x <- model_3pl()$gendata(10, 5)
irt_select(x, people_index=c(1, 3, 5))
irt_select(x, item_index=c(1, 3, 5))
# sample wihtout replacement
irt_sample(x, n.people=3)
irt_sample(x, n.items=3)
# sample with replacement
irt_sample(x, n.people=30)
irt_sample(x, n.items=30)
# rescale theta
x <- model_3pl()$gendata(20, 5)
c(mean(x$people$theta), sd(x$people$theta))
y <- irt_rescale(x, "theta", 0, 1)
```

```
c(mean(y$people$theta), sd(y$people$theta))
round(abs(irt_stats(x, "prob") - irt_stats(y, "prob")), 2)
# rescale b
c(mean(x$items$b), sd(x$items$b))
y <- irt_rescale(x, "b", 0, 1)
c(mean(y$items$b), sd(y$items$b))
round(abs(irt_stats(x, "prob") - irt_stats(y, "prob")), 2)
```

`model_3pl` *3-Parameter-Logistic Model*

Description

Create a 3-parameter-logistic (3PL) model object

Usage

```
model_3pl(people = NULL, items = NULL, responses = NULL, theta = NULL,
  a = NULL, b = NULL, c = NULL)

## S3 method for class 'model.3pl'
print(x, ...)

## S3 method for class 'model.3pl'
plot(x, ...)
```

Arguments

`people`	people parameters (data.frame)
`items`	item parameters (data.frame)
`responses`	dichotomous responses (data.frame or matrix)
`theta`	the ability parameters (vector)
`a`	the discrimination parameters (vector)
`b`	the difficulty parameters (vector)
`c`	the pseudo-guessing parameters (vector)
`x`	a `model.3pl` object
`...`	additional arguments

Details

A 3pl model contains `people` (people parameters), `items` (item parameters), `responses` (responses data), and functions to compute `P` (probability), `I` (information), and `L` (likelihood).
Arguments are allowed to be `NULL`. The `people` argument needs to have a column named `theta`. The `items` argument needs to have columns named `a`, `b`, and `c`. The `responses` argument needs to be a data frame or matrix whose dimensionality matches with `people` and `items`.

In `plot.model.3pl`, use the `stats` argument to control what IRT statistics to draw. Use `total` argument to control whether to aggregate results over items or people.

Examples

```
# create a 3pl model using given parameters
theta <- c(-1, 0, 1)
a <- c(.5882, 1)
b <- c(-1, 1)
c <- c(0, .2)
u <- matrix(c(1, 0, 1, 0, 1, 0), nrow=3)
people <- data.frame(theta=theta)
items <- data.frame(a=a, b=b, c=c)
model_3pl(people=people, items=items, responses=u)
model_3pl(people=people, items=items)
model_3pl(theta=theta, a=a, b=b, c=c)
model_3pl(people=people, a=a, b=b, c=c)
model_3pl(theta=theta, items=items)
# compute P(robability), I(nformation), L(ikelihood)
x <- model_3pl(people=people, items=items, responses=u)
x$P()
x$I()
x$L()
model_3pl()$P(x)
model_3pl()$I(x)
model_3pl()$L(x)
# create a 3pl model using generated data
x <- model_3pl()$gendata(10, 5)
x$P(x)
x$I(x)
x$L(x)
# draw test/item characteristic curve
x <- model_3pl()$gendata(20, 5)
plot(x, stats="prob")
plot(x, stats="prob", total=FALSE)
# draw test/iten information function
plot(x, stats="info")
plot(x, stats="info", total=FALSE)
# draw loglikelihood
plot(x, stats="loglik")
plot(x, stats="loglik", total=FALSE, theta=seq(-5, 5, .1))
```

`mst` *Computerized Multistage Testing (MST)*

Description

`mst` creates a multistage (MST) object

`mst_route` adds and removes routes in the MST

`mst_obj` adds objective functions to the MST

`mst_constraint` adds constraints to the MST

`mst_stage_length` sets the minimum and maximum length for a stage

`mst_rdp` anchors the routing decision point (rdp) between adjacent modules

`mst_module_mininfo` sets the minimum information for modules

`mst_assemble` assembles the mst

`mst_get_items` extracts items from results

Usage

```
mst(pool, design, npanel, method = c("topdown", "bottomup"), len = NULL,
  maxselect = NULL)

mst_route(x, route, op = c("+", "-"))

mst_obj(x, theta, indices = NULL, target = NULL, flatten = NULL)

mst_constraint(x, coef, min = NA, max = NA, level = NULL,
  indices = NULL)

mst_stage_length(x, stages, min = NA, max = NA)

mst_rdp(x, theta, indices, tol)

mst_module_mininfo(x, theta, mininfo, indices)

mst_assemble(x, ...)

## S3 method for class 'mst'
print(x, ...)

## S3 method for class 'mst'
plot(x, ...)

mst_get_items(x, panel, stage = NULL, module = NULL, route = NULL)
```

Arguments

`pool`	the item pool (data.frame)
`design`	the MST design (string): e.g., "1-2-3", "1-2-2", etc.
`npanel`	the number of panels
`method`	the design method: 'topdown' or 'bottomup'
`len`	the module/route length
`maxselect`	the maximum selection of items
`x`	a MST object
`route`	a MST route represented by a vector of module indices
`op`	`"+"` for adding a route and `"-"` for removing a route
`theta`	a theta point or interval for optimization

indices	the index of the route (topdown) or the module (bottomup) for adding objectives
target	the target values of the TIF objectives. NULL for maximization
flatten	the parameter for getting a flat TIF
coef	the coefficients of the constraint
min	the lower bound of the constraints
max	the upper bound of the constraints
level	the constrained level, NA for continuous variable
stages	the stage index
tol	tolerance parameter
mininfo	the minimum information threshold
...	further arguments
panel	the panel index
stage	the stage index
module	the module index

Details

The mst object contains: item pool (pool), ATA (assembler), route map (route), module map (module), design method (method), and several constants such as npanel, nstage, nmodule, nroute. Two identifiers are used to index modules/testlets. form is a unique id for all modules (for ATA), and index is a panel-wise unique id (for MST).
There are two methods for designing MST: 'bottomup' and 'topdown'. The bottomup approach adds objectives and constraints on individual modules, whereas the topdown approach adds objectives and constraints directly on routes.

plot.mst draws module information functions if byroute=FALSE and route information functions if byroute=TRUE

Examples

```
## Not run:
library(dplyr)
set.seed(10001)
## generate item pool
pool <- model_3pl()$gendata(1, 300)$items
pool$content <- sample(1:3, nrow(pool), replace=TRUE)
pool$time <- round(exp(rnorm(nrow(pool), log(60), .2)))

## ex. 1: 1-2-2 MST, 2 panels, topdown
## 20 items in total, 10 items in content area 1
## maximize info. at -1 and 1 for easy and hard routes
x <- mst(pool, "1-2-2", 2, 'topdown', len=20, maxselect=1)
x <- mst_obj(x, theta=-1, indices=1:2)
x <- mst_obj(x, theta=1, indices=3:4)
x <- mst_constraint(x, "content", 10, 10, level=1)
x <- mst_assemble(x, timeout=10)
```

```
plot(x)
plot(x, byroute=TRUE)
freq(mst_get_items(x, panel=1, route=1)$content, 1:3)$freq
freq(mst_get_items(x, panel=2, route=4)$content, 1:3)$freq

## ex. 2: 1-2-3 MST, 2 panels, bottomup,
## 10 items in total and 4 items in content area 1 in each module
## maximize info. at -1, 0 and 1 for easy, medium, and hard modules
x <- mst(pool, "1-2-3", 2, 'bottomup', len=10, maxselect=1) %>%
  mst_route(c(1, 2, 6), "-") %>%
  mst_route(c(1, 3, 4), "-") %>%
  mst_obj(theta= 0, indices=c(1, 5)) %>%
  mst_obj(theta=-1, indices=c(2, 4)) %>%
  mst_obj(theta= 1, indices=c(3, 6)) %>%
  mst_constraint("content", 4, 4, level=1)
x <- mst_assemble(x, timeout=10)
group_by(x$items, panel, index) %>% summarise(n=sum(content==1))

## End(Not run)
```

`mstHelpers` *MST Helper Functions*

Description

MST Helper Functions

`mst_reindex_routes` sorts an re-indexes the route map

`mst_indices_input` converts indices input to indices of routes in topdown method or modules in bottomup method

Usage

```
mst_reindex_routes(x)

mst_indices_input(x, indices)
```

Arguments

`x`	a `mst` object
`indices`	the route or module indices

mst_sim *MST Simulation*

Description

mst_sim runs a MST simulation

Usage

```
mst_sim(x, true, rdp = NULL, ...)

## S3 method for class 'mst.sim'
print(x, ...)

## S3 method for class 'mst.sim'
plot(x, ...)
```

Arguments

x	the assembled MST
true	the true theta parameter
rdp	a list of routing decision points
...	additional option/control parameters

Examples

```
## Not run:
set.seed(10001)
pool <- model_3pl()$gendata(1,200)$items
pool$content <- sample(1:3, nrow(pool), replace=TRUE)
pool$time <- round(rlnorm(nrow(pool), log(60), .2))
x <- mst(pool, "1-2", 2, 'topdown', len=20, maxselect=1)
x <- mst_obj(x, theta=-1, indices=1)
x <- mst_obj(x, theta= 1, indices=2)
x <- mst_constraint(x, "content", 10, 10, level=1)
x <- mst_stage_length(x, 1, min=5)
x <- mst_assemble(x, timeout=10)
rdp <- list(stage1=0)
mst_sim(x, 1.0, rdp)

## End(Not run)
```

`rmse` *Root Mean Square Error*

Description

compute the root mean squared error of two numeric vectors/matrices

Usage

```
rmse(x, y)
```

Arguments

`x`	a vector/matrix of numeric values
`y`	a vector/matrix of numeric values

Index