Package 'xxIRT'

July 6, 2018

Type Package

Title Item Response Theory and Computer-Based Testing in R

Version 2.1.0

Date 2018-7-11

Author Xiao Luo [aut, cre]

Maintainer Xiao Luo <xluo1986@gmail.com>

Description A suite of practical psychometric analysis and research tools of item response theory, including the 3-parameter-logistic model and the generalized partial credit model as well as functions to perform: (1) parameter estimation, (2) automated test assembly, (3) simulation of computerized adaptive testing (CAT), and (4) assembly and simulation of multistage testing. 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 ggplot2, lpSolveAPI, reshape2, stats, utils

RoxygenNote 6.0.1

Suggests testthat

NeedsCompilation no

Repository CRAN

Date/Publication 2018-07-05 22:50:02 UTC

R topics documented:

---

ata *Automated Test Assembly (ATA)*

Description

Automated Test Assembly (ATA)

`ata` creates an `ata` object

`ata_append_constraints` appends constraint definitions to the ata object

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

`ata_obj_coef` processes input coefficients of the objective functions

`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_use` sets the minimum and maximum use 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 object

`ata_solve_lpsolve` solves the LP using LpSolve

Usage

```
ata(pool, num_form = 1, len = NULL, max_use = NULL, group = NULL, ...)

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

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

ata_append_constraints(x, mat, dir, rhs)

ata_form_index(x, forms, collapse)

ata_obj_coef(x, coef, compensate)

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_use(x, min = NA, max = NA, 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 (must include a, b, c parameters), a data.frame
num_form	the number of forms to be assembled
len	the length of each form
max_use	the maximum use of each item
group	the grouping indices
...	further arguments
x	the ata object
mat	the coefficient matrix
dir	direction
rhs	right-hand-side value
forms	the indices of forms where objectives are added. NULL for all forms
collapse	TRUE to collapse all forms into one objective function
coef	the coefficients of the objective function or the constraint
compensate	TRUE to combine coefficients
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
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.

Examples

```
## Not run:
## generate a 100-item pool
nitems <- 100
pool <- with(model_3pl_gendata(1, nitems), data.frame(id=1:nitems, a=a, b=b, c=c))
pool$id <- 1:nitems
pool$content <- sample(1:3, nitems, replace=TRUE)
pool$time <- round(rlnorm(nitems, log(60), .2))
pool$group <- sort(sample(1:round(nitems/3), nitems, replace=TRUE))

## ex. 1: 6 forms, 10 items, maximize b parameter
x <- ata(pool, 6, len=10, max_use=1)
x <- ata_obj_relative(x, "b", "max")
x <- ata_solve(x, timeout=5)
sapply(x$items, function(x) summary(x$b))

## ex. 2: 3 forms, 10 items, minimize b parameter
x <- ata(pool, 3, len=10, max_use=1)
x <- ata_obj_relative(x, "b", "min", negative=TRUE)
x <- ata_solve(x, as.list=FALSE, timeout=5)
aggregate(x$items$b, by=list(form=x$items$form), summary)

## ex. 3: 2 forms, 10 items, mean(b) = 0, sd(b) = 1.0, content = (3, 3, 4)
x <- ata(pool, 2, len=10, max_use=1)
x <- ata_obj_absolute(x, pool$b, 0 * 10)
x <- ata_obj_absolute(x, (pool$b - 0)^2, 1 * 10)
x <- ata_constraint(x, "content", min=3, max=3, level=1)
x <- ata_constraint(x, "content", min=3, max=3, level=2)
x <- ata_constraint(x, "content", min=4, max=4, level=3)
x <- ata_solve(x, timeout=5)
sapply(x$items, function(x) c(mean=mean(x$b), sd=sd(x$b)))

## ex. 4: same with ex. 3, but group-based
```

```
x <- ata(pool, 2, len=10, max_use=1, group="group")
x <- ata_obj_absolute(x, pool$b, 0 * 10)
x <- ata_obj_absolute(x, (pool$b - 0)^2, 1 * 10)
x <- ata_constraint(x, "content", min=3, max=3, level=1)
x <- ata_constraint(x, "content", min=3, max=3, level=2)
x <- ata_constraint(x, "content", min=4, max=4, level=3)
x <- ata_solve(x, timeout=5)
sapply(x$items, function(x) c(mean=mean(x$b), sd=sd(x$b),
       n_items=length(unique(x$id)), n_groups=length(unique(x$group))))

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

## End(Not run)
```

`cat_sim` *Simulation of Computerized Adaptive Testing (CAT)*

Description

Simulation of Computerized Adaptive Testing (CAT)

`cat_sim` uns a simulation of CAT. Use `theta` in options to set the starting value of theta estimate.

`cat_estimate_mle` is the maximum likelihood estimation rule. Use `map_len` to apply MAP to the first K items and use `map_prior` to set the prior for MAP.

`cat_estimate_eap` is the expected a posteriori estimation rule, using `eap_mean` and `eap_sd` option parameters as the prior

`cat_estimate_hybrid` is a hybrid estimation rule, which uses MLE for mixed responses and EAP for all 1's or 0's responses

`cat_stop_default` is a three-way stopping rule. When `stop_se` is set in the options, it uses the standard error stopping rule. When `stop_mi` is set in the options, it uses the minimum information stopping rule. When `stop_cut` is set in the options, it uses the confidence interval (set by `ci_width`) stopping rule.

`cat_select_maxinfo` is the maximum information selection rule. Use `group` (a numeric vector) to group items belonging to the same set. Use `info_random` to implement the random-esque item exposure control method.

`cat_select_ccat` is the constrained CAT selection rule. Use `ccat_var` to set the content variable in the pool. Use `ccat_perc` to set the desired content distribution, with the name of each element being the content code and tue value of each element being the percentage. Use `ccat_random` to add randomness to initial item selections.

`cat_select_shadow` is the shadow-test selection rule. Use `shadow_id` to group item sets. Use `constraints` to set constraints. Constraints should be in a data.frame with four columns: var (variable name), level (variable level, `NA` for quantitative variable), min (lower bound), and max (upper bound).

cat_stop_projection is the projection-based stopping rule. Use projection_method to choose the projection method ('info' or 'diff'). Use stop_cut to set the cut score. Use constraints to set the constraints. Constraints should be a data.frame with columns: var (variable name), level (variable level, NA for quantitative varialbe), min (lower bound), max (upper bound)

Usage

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

cat_estimate_mle(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_maxinfo(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, ...)

cat_stop_projection(len, theta, stats, admin, pool, opts)
```

Arguments

true	the true theta
pool	the item pool (data.frame)
...	option/control parameters
len	the current test length
theta	the current theta estimate
stats	a matrix of responses, theta estimate, information and std error
admin	a data frame of administered items
opts	a list of option/control parameters
x	a cat object

Details

... takes a variety of option/control parameters for the simulations from users. min and max are mandatory for setting limits on the test length. User-defined selection, estimation, and stopping

rules are also passed to the simulator via options.
To write a new rule, the function siganiture must be: `function(len, theta, stats, admin, pool, opts)`.
See built-in rules for examples.

Value

`cat_sim` returns a `cat` object

an estimation rule should return a theta estimate

a stopping rule should return a boolean: `TRUE` to stop the CAT, `FALSE` to continue

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

Examples

```
## Not run:
## generate a 100-item pool
num_items <- 100
pool <- with(model_3pl_gendata(1, num_items), data.frame(a=a, b=b, c=c))
pool$set_id <- sample(1:30, num_items, replace=TRUE)
pool$content <- sample(1:3, num_items, replace=TRUE)
pool$time <- round(rlnorm(num_items, mean=4.1, sd=.2))

## MLE, EAP, and hybrid estimation rule
cat_sim(1.0, pool, min=10, max=20, estimate_rule=cat_estimate_mle)
cat_sim(1.0, pool, min=10, max=20, estimate_rule=cat_estimate_eap)
cat_sim(1.0, pool, min=10, max=20, estimate_rule=cat_estimate_hybrid)

## SE, MI, and CI stopping rule
cat_sim(1.0, pool, min=10, max=20, stop_se=.3)
cat_sim(1.0, pool, min=10, max=20, stop_mi=.6)
cat_sim(1.0, pool, min=10, max=20, stop_cut=0)
cat_sim(1.0, pool, min=10, max=20, stop_cut=0, ci_width=2.58)

## maximum information selection with item sets
cat_sim(1.0, pool, min=10, max=20, group="set_id")$admin

## maximum information with item exposure control
cat_sim(1.0, pool, min=10, max=20, info_random=5)$admin

## Constrained-CAT selection rule with and without 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))
cat_sim(1.0, pool, min=10, max=20, select_rule=cat_select_ccat, ccat_random=5,
        ccat_var="content", ccat_perc=c("1"=.2, "2"=.3, "3"=.5))

## Shadow-test selection rule
cons <- data.frame(var='content', level=1:3, min=c(3,3,4), max=c(3,3,4))
cons <- rbind(cons, data.frame(var='time', level=NA, min=55*10, max=65*10))
cat_sim(1.0, pool, min=10, max=10, select_rule=cat_select_shadow, constraints=cons)

## Projection-based stopping rule
cons <- data.frame(var='content', level=1:3, min=5, max=15)
```

```
cons <- rbind(cons, data.frame(var='time', level=NA, min=60*20, max=60*40))
cat_sim(1.0, pool, min=20, max=40, select_rule=cat_select_shadow, stop_rule=cat_stop_projection,
        projection_method="diff", stop_cut=0, constraints=cons)

## End(Not run)
```

`model_3pl` *3-parameter-logistic model*

Description

3-parameter-logistic model

`model_3pl_prob` computes the probability of a correct response for given parameters

`model_3pl_info` computes the information for given parameters

`model_3pl_lik` computes the likelihood of observed responses

`model_3pl_rescale` transforms parameterrs to a given scale

`model_3pl_gendata` generates data using the 3pl model

`model_3pl_plot` plots the item characteristics curve (ICC) or item information function curve (IIFC)

`model_3pl_plot_loglik` plots the log-likelihood curves for each response vector

`model_3pl_jmle` estimates parameters using the joint MLE method

`model_3pl_mmle` estimates parameters using the marginal MLE method

`model_3pl_eap_scoring` computes scores using the EAP method

Usage

```
model_3pl_prob(t, a, b, c, D = 1.702)

model_3pl_info(t, a, b, c, D = 1.702)

model_3pl_lik(u, t, a, b, c, D = 1.702, log = FALSE)

model_3pl_rescale(t, a, b, c, param = c("t", "b"), mean = 0, sd = 1)

model_3pl_gendata(num_people, num_item, t = NULL, a = NULL, b = NULL,
  c = NULL, D = 1.702, t_dist = c(0, 1), a_dist = c(0, 0.2),
  b_dist = c(0, 1), c_dist = c(5, 46), missing = NULL)

model_3pl_plot(a, b, c, D = 1.702, type = c("prob", "info"),
  total = FALSE, xaxis = seq(-4, 4, 0.1))

model_3pl_plot_loglik(u, a, b, c, D = 1.702, xaxis = seq(-4, 4, 0.1),
  show_mle = FALSE)
```

```
model_3pl_dv_t(u, t, a, b, c, D, prior = NULL)

model_3pl_dv_a(u, t, a, b, c, D, prior = NULL, post_prob = NULL)

model_3pl_dv_b(u, t, a, b, c, D, prior = NULL, post_prob = NULL)

model_3pl_dv_c(u, t, a, b, c, D, prior = NULL, post_prob = NULL)

model_3pl_estimate_inits(u, t, a, b, c)

model_3pl_estimate_nr(param, h, is_free, h_max, bounds)

model_3pl_jmle(u, t = NA, a = NA, b = NA, c = NA, D = 1.702,
  num_iter = 100, num_nr = 15, h_max = 1, conv = 0.1, decay = 0.95,
  scale = NULL, bounds = list(t = c(-4, 4), a = c(0.1, 2), b = c(-4, 4), c =
  c(0.01, 0.3)), priors = list(t = c(0, 1), a = c(0, 0.2), b = c(0, 1), c =
  c(5, 46)), debug = FALSE)

model_3pl_posterior_dist(u, a, b, c, D, quad_t, quad_w)

model_3pl_mmle(u, a = NA, b = NA, c = NA, D = 1.702, num_iter = 100,
  num_nr = 15, num_quad = c("11", "20"), h_max = 1, conv = 0.1,
  decay = 0.98, scale = NULL, bounds = list(t = c(-4, 4), a = c(0.1, 2), b
  = c(-4, 4), c = c(0.01, 0.3)), priors = list(t = c(0, 1), a = c(0, 0.2), b =
  c(0, 1), c = c(5, 46)), debug = FALSE)

model_3pl_eap_scoring(u, a, b, c, D)
```

Arguments

t	ability parameter, 1d vector
a	discrimination parameters, 1d vector
b	difficulty parameters, 1d vector
c	guessing parameters, 1d vector
D	the scaling constant, 1.702 by default
u	observed responses, 2d matrix
log	True to return log-likelihood
param	the parameter of the new scale, t or b
mean	the mean of the new scale
sd	the SD of the new scale
num_people	the number of people to be generated
num_item	the number of items to be generated
t_dist	the normal distribution parameters of t-parameters, a vector: c(mean, sd)
a_dist	the lognormal distribution parameters of a-parameters, a vector: c(meanlog, sd-log)

b_dist	the normal distribution parameters of b-parameters, a vector: c(mean, sd)
c_dist	the beta distribution pamameters of c-parameters, a vector: c(alpha, beta)
missing	the proportion or number of missing responses
type	the type of plot, prob for ICC and info for IIFC
total	TRUE to sum values over items
xaxis	the values of x-axis
show_mle	TRUE to print maximum likelihood values
prior	parameters of the prior distribution
post_prob	posterior distribution of the theta
h	change of parameters in the newton-raphson method
is_free	TRUE to estimate parameters and FALSE to fix parameters
h_max	the maximum value of h in the newton-raphson method
bounds	the bounds of parameters, a list
num_iter	the maximum number of overall iterations
num_nr	the maximum number of the newton-raphson iterations
conv	the convergence criterion in -2 log-likelihood of model fit
decay	the epoch decay parameter
scale	the scale of theta parameters
priors	the priors of parameters used in the maximum a posteriori estimation
debug	TRUE to print debugging information
quad_t	values of quadrature points
quad_w	weights of quadrature points
num_quad	the number of quadrature points

Value

`model_3pl_prob` returns a 2d matrix of probabilities, dim = n_people x n_items

`model_3pl_info` returns a 2d matrix of informations, dim = n_people x n_items

`model_3pl_lik` returns a 2d matrix of likelihood, dim = n_people x n_items

`model_3pl_rescale` returns a list of rescaled t, a, b, c parameters

`model_3pl_gendata` returns a list of generated data

`model_3pl_plot` returns a ggplot graph

`model_3pl_plot` returns a ggplot graph

`model_3pl_jmle` returns estimated parameters

`model_3pl_eap_scoring` returns the EAP scores

Examples

```
model_3pl_prob(c(-1, 1), c(1, .8, .8), c(-1, 1, 1), c(0, 0, .2), 1.702)
model_3pl_prob(c(-1, 1), c(1, .8, .8), c(-1, 1, 1), c(0, 0, .2), 1.0)
model_3pl_info(c(-1, 1), c(1, .8, .8), c(-1, 1, 1), c(0, 0, .2), 1.702)
model_3pl_info(c(-1, 1), c(1, .8, .8), c(-1, 1, 1), c(0, 0, .2), 1.0)
u <- matrix(c(1, 0, 1, 0, 1, 0), nrow=2)
model_3pl_lik(u, c(-1, 1), c(1, .8, .8), c(-1, 1, 1), c(0, 0, .2), 1.702, FALSE)
model_3pl_lik(u, c(-1, 1), c(1, .8, .8), c(-1, 1, 1), c(0, 0, .2), 1.0, TRUE)
model_3pl_gendata(10, 5)
model_3pl_gendata(10, 5, a=1, c=0, missing=.1)
with(model_3pl_gendata(10, 5), model_3pl_plot(a, b, c, type="prob"))
with(model_3pl_gendata(10, 5), model_3pl_plot(a, b, c, type="info", total=TRUE))
with(model_3pl_gendata(5, 50), model_3pl_plot_loglik(u, a, b, c, show_mle=TRUE))
## Not run:
data_tru <- model_3pl_gendata(3000, 50)
data_est <- model_3pl_jmle(u=data_tru$u, scale=c(0, 1), priors=NULL, debug=TRUE)
evaluate_3pl_estimation(data_tru, data_est)

## End(Not run)
## Not run:
data_tru <- model_3pl_gendata(3000, 50)
data_est <- model_3pl_mmle(u=data_tru$u, scale=NULL, priors=NULL, debug=TRUE)
evaluate_3pl_estimation(data_tru, data_est)

## End(Not run)
```

`model_gpcm` *Generalized partial credit model*

Description

Generalized partial credit model

`model_gpcm_prob` computes the probability of all score categories

`model_gpcm_info` computes the informations of all score categories

`model_gpcm_onehot_response` converts 2-dimensional score matrix to a 3-dimensional one-hot vector of score category

`model_gpcm_lik` computes the (log-)likelihood of the responses

`model_gpcm_gendata` generates data using the GPCM model

`model_gpcm_plot` plots the item characteristics curve (ICC) or item information function curve (IIFC)

`model_gpcm_plot_loglik` plots the log-likelihood curves for each response vector

`model_gpcm_jmle` estimates the parameters using the joint MLE method

Usage

```
model_gpcm_prob(t, a, b, d, D = 1.702, add_initial = NULL)

model_gpcm_info(t, a, b, d, D = 1.702, add_initial = NULL)

model_gpcm_onehot_response(u, num_category = NULL)

model_gpcm_extract_3Ddata(data, u)

model_gpcm_lik(u, t, a, b, d, D = 1.702, add_initial = NULL, log = FALSE)

model_gpcm_gendata(num_people, num_item, num_category, sort_b = TRUE,
  t = NULL, a = NULL, b = NULL, D = 1.702, set_initial = NULL,
  t_dist = c(0, 1), a_dist = c(0, 0.2), b_dist = c(0, 1),
  missing = NULL)

model_gpcm_plot(a, b, d, D = 1.702, add_initial = NULL, type = c("prob",
  "info"), by_item = FALSE, total = FALSE, xaxis = seq(-6, 6, 0.1))

model_gpcm_plot_loglik(u, a, b, d, D = 1.702, add_initial = NULL,
  xaxis = seq(-6, 6, 0.1), show_mle = FALSE)

model_gpcm_dv_t(u, t, a, b, D, prior = NULL)

model_gpcm_dv_a(u, t, a, b, D, prior = NULL, post_prob = NULL)

model_gpcm_dv_b(u, t, a, b, D, prior = NULL, post_prob = NULL)

model_gpcm_estimate_inits(u, t, a, b, d, set_initial)

model_gpcm_estimate_nr(param, h, is_free, h_max, bounds)

model_gpcm_jmle(u, t = NA, a = NA, b = NA, d = NA, D = 1.702,
  set_initial = 0, num_iter = 100, num_nr = 15, h_max = 1, conv = 0.1,
  decay = 0.95, scale = NULL, bounds = list(t = c(-4, 4), a = c(0.1, 2), b
  = c(-4, 4)), priors = list(t = c(0, 1), a = c(0, 0.2), b = c(0, 1)),
  debug = FALSE)
```

Arguments

t	ability parameters, a vector
a	discrimination parameters, a vector
b	item location parameters when it's a matrix; item-category parameters when it's a vector
d	NULL or item category parameters
D	the scaling constant
add_initial	the initial value added to item category parameters

u	the observed scores (starting from 0), 2d matrix
num_category	the number of score categories
data	the 3-dimensional data
log	TRUE to return log-likelihood
num_people	the number of people to be generated
num_item	the number of items to be generated
sort_b	TRUE to sort b parameters of each item
set_initial	the value of the initial category
t_dist	the normal distribution parameters of t-parameters, a vector: c(mean, sd)
a_dist	the lognormal distribution parameters of a-parameters, a vector: c(meanlog, sd-log)
b_dist	the normal distribution parameters of b-parameters, a vector: c(mean, sd)
missing	the proportion or number of missing responses
type	the type of plot, prob for ICC and info for IIFC
by_item	TRUE to combine categories
total	TRUE to sum values over items
xaxis	the values of x-axis
show_mle	TRUE to print maximum likelihood values
prior	parameters of the prior distribution
post_prob	posterior distribution of the theta
param	the parameters to be updated
h	change of parameters in the newton-raphson method
is_free	TRUE to estimate parameters and FALSE to fix parameters
h_max	the maximum value of h in the newton-raphson method
bounds	the bounds of parameters, a list
num_iter	the maximum number of overall iterations
num_nr	the maximum number of the newton-raphson iterations
conv	the convergence criterion in -2 log-likelihood of model fit
decay	the epoch decay parameter
scale	the scale of theta parameters
priors	the priors of parameters used in the maximum a posteriori estimation
debug	TRUE to print debugging information

Details

The GPCM has two kinds of parameterization: (1) b as the item-category parameters or (2) b - d as the item-category parameters. In (1), b must be a matrix and d NULL. In (2), b must be a vector and d a matrix. Use NA to represent unused category. Use initial to insert the values of the initial category.

Value

`model_gpcm_gendata` returns a list of generated data

`model_gpcm_plot` returns a ggplot graph

`model_gpcm_plot` returns a ggplot graph

Examples

```
model_gpcm_gendata(10, 5, 3)
model_gpcm_gendata(10, 5, 3, set_initial=0, missing=.1)
# Figure 1 in Muraki, 1992 (APM)
model_gpcm_plot(a=c(1,1,.7), b=matrix(c(-2,0,2,-.5,0,2,-.5,0,2), nrow=3, byrow=TRUE),
    d=NULL, D=1.0, add_initial=0, xaxis=seq(-4, 4, .1), type='prob')
# Figure 2 in Muraki, 1992 (APM)
model_gpcm_plot(a=.7, b=matrix(c(.5,0,NA,0,0,0), nrow=2, byrow=TRUE),
    d=NULL, D=1.0, add_initial=0, xaxis=seq(-4, 4, .1))
# Figure 3 in Muraki, 1992 (APM)
model_gpcm_plot(a=c(.778,.946), b=matrix(c(1.759,-1.643,3.970,-2.764), nrow=2, byrow=TRUE),
    d=NULL, D=1.0, add_initial=0)
# Figure 1 in Muraki, 1993 (APM)
model_gpcm_plot(a=1, b=matrix(c(0,-2,4,0,-2,2,0,-2,0,0,-2,-2,0,-2,-4), nrow=5, byrow=TRUE),
    d=NULL, D=1.0)
# Figure 2 in Muraki, 1993 (APM)
model_gpcm_plot(a=1, b=matrix(c(0,-2,4,0,-2,2,0,-2,0,0,-2,-2,0,-2,-4), nrow=5, byrow=TRUE),
    d=NULL, D=1.0, type='info', by_item=TRUE)
with(model_gpcm_gendata(5, 50, 3, set_initial=0),
    model_gpcm_plot_loglik(u, a, b, NULL, show_mle=TRUE))
## Not run:
data_tru <- model_gpcm_gendata(1500, 30, 3, set_initial=0)
data_est <- model_gpcm_jmle(data_tru$u, prior=NULL, debug=TRUE)
evaluate_gpcm_estimation(data_tru, data_est)

## End(Not run)
```

`mst` *Computerized Multistage Testing (MST)*

Description

Computerized Multistage Testing (MST)

`mst` creates a multistage (MST) object for assembly

`mst_route` adds/removes a route to/from the MST

`mst_get_indices` maps the input indices to the actual indices

`mst_obj` adds objective functions to the MST

`mst_constraint` adds constraints to the MST

`mst_stage_length` sets length limits on stages

`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 the assembly results

Usage

```
mst(pool, design, num_panel, method = c("topdown", "bottomup"), len = NULL,
  max_use = NULL, group = NULL, ...)

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

mst_get_indices(x, indices)

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 = NULL, stage = NULL, module = NULL,
  route = NULL, route_index = NULL)
```

Arguments

`pool`	the item pool (data.frame)
`design`	the MST design (string): e.g., "1-3", "1-2-2", "1-2-3"
`num_panel`	the number of panels (integer)
`method`	the design method (string): 'topdown' or 'bottomup'
`len`	the module/route length (integer)
`max_use`	the maximum selection of items (integer)
`group`	the grouping variable (string or vector)
`...`	further arguments
`x`	the MST object

route	a MST route represented by a vector of module indices
op	"+" to add a route and "-" to remove a route
indices	the indices of the route (topdown) or the module (bottomup) where objectives are added
theta	a theta point or interval over which the TIF is optimized
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 constraint
max	the upper bound of the constraint
level	the constrained level, NA for quantitative variable
stages	the stage indices
tol	tolerance parameter (numeric)
mininfo	the minimum information threshold
panel	the panel indices (numeric vector)
stage	the stage indices (numeric vector)
module	the module indices (numeric vector)
route_index	the route indices (integer)

Details

There are two methods for designing a MST. The bottom-up 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 when byroute=FALSE and route information functions when byroute=TRUE

Examples

```
## Not run:
## generate item pool
num_item <- 300
pool <- with(model_3pl_gendata(1, num_item), data.frame(a=a, b=b, c=c))
pool$id <- 1:num_item
pool$content <- sample(1:3, num_item, replace=TRUE)
pool$time <- round(rlnorm(num_item, 4, .3))
pool$group <- sort(sample(1:round(num_item/3), num_item, replace=TRUE))

## ex. 1: 1-2-2 MST, 2 panels, topdown
## 20 items in total and 10 items in content area 1 in each route
## maximize info. at -1 and 1 for easy and hard routes
x <- mst(pool, "1-2-2", 2, 'topdown', len=20, max_use=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=5)
plot(x, byroute=TRUE)
for(p in 1:x$num_panel)
  for(r in 1:x$num_route) {
     route <- paste(x$route[r, 1:x$num_stage], collapse='-')
     count <- sum(mst_get_items(x, panel=p, route_index=r)$content==1)
    cat('panel=', p, ', route=', route, ': ', count, ' items in content area #1\n', sep='')
  }

## ex. 2: 1-2-3 MST, 2 panels, bottomup,
## remove two routes with large theta change: 1-2-6, 1-3-4
## 10 items in total and 4 items in content area 2 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, max_use=1)
x <- mst_route(x, c(1, 2, 6), "-")
x <- mst_route(x, c(1, 3, 4), "-")
x <- mst_obj(x, theta= 0, indices=c(1, 5))
x <- mst_obj(x, theta=-1, indices=c(2, 4))
x <- mst_obj(x, theta= 1, indices=c(3, 6))
x <- mst_constraint(x, "content", 4, 4, level=2)
x <- mst_assemble(x, timeout=10)
plot(x, byroute=FALSE)
for(p in 1:x$num_panel)
  for(m in 1:x$num_module){
    count <- sum(mst_get_items(x, panel=p, module=m)$content==2)
    cat('panel=', p, ', module=', m, ': ', count, ' items in content area #2\n', sep='')
  }

## ex.3: same with ex.2 (w/o content constraints), but group-based
x <- mst(pool, "1-2-3", 2, 'bottomup', len=12, max_use=1, group="group")
x <- mst_route(x, c(1, 2, 6), "-")
x <- mst_route(x, c(1, 3, 4), "-")
x <- mst_obj(x, theta= 0, indices=c(1, 5))
x <- mst_obj(x, theta=-1, indices=c(2, 4))
x <- mst_obj(x, theta= 1, indices=c(3, 6))
x <- mst_assemble(x, timeout=10)
plot(x, byroute=FALSE)
for(p in 1:x$num_panel)
  for(m in 1:x$num_module){
    items <- mst_get_items(x, panel=p, module=m)
    cat('panel=', p, ', module=', m, ': ', length(unique(items$id)), ' items from ',
        length(unique(items$group)), ' groups\n', sep='')
  }

## ex.4: 2 panels of 1-2-3 top-down design
## 20 total items and 10 items in content area 3
## 6+ items in stage 1 & 2
x <- mst(pool, "1-2-3", 2, "topdown", len=20, max_use=1)
x <- mst_route(x, c(1, 2, 6), "-")
x <- mst_route(x, c(1, 3, 4), "-")
x <- mst_obj(x, theta=-1, indices=1)
x <- mst_obj(x, theta=0, indices=2:3)
x <- mst_obj(x, theta=1, indices=4)
```

```
x <- mst_constraint(x, "content", 10, 10, level=3)
x <- mst_stage_length(x, 1:2, min=6)
x <- mst_assemble(x, timeout=15)
head(x$items)
plot(x, byroute=FALSE)
for(p in 1:x$num_panel)
  for(s in 1:x$num_stage){
    items <- mst_get_items(x, panel=p, stage=s)
    cat('panel=', p, ', stage=', s, ': ', length(unique(items$id)), ' items\n', sep='')
  }

## ex.5: same with ex.4, but use RDP instead of stage length to control routing errors
x <- mst(pool, "1-2-3", 2, "topdown", len=20, max_use=1)
x <- mst_route(x, c(1, 2, 6), "-")
x <- mst_route(x, c(1, 3, 4), "-")
x <- mst_obj(x, theta=-1, indices=1)
x <- mst_obj(x, theta=0, indices=2:3)
x <- mst_obj(x, theta=1, indices=4)
x <- mst_constraint(x, "content", 10, 10, level=3)
x <- mst_rdp(x, 0, 2:3, .1)
x <- mst_module_mininfo(x, 0, 5, 2:3)
x <- mst_assemble(x, timeout=15)
plot(x, byroute=FALSE)

## End(Not run)
```

`mst_sim` *Simulation of Multistage Testing*

Description

Simulation of Multistage Testing

`mst_sim` simulates a MST administration

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 (numeric)
`rdp`	routing decision points (list)
`...`	additional option/control parameters

Examples

```
## Not run:
## assemble a MST
nitems <- 200
pool <- with(model_3pl_gendata(1, nitems), data.frame(a=a, b=b, c=c))
pool$content <- sample(1:3, nrow(pool), replace=TRUE)
x <- mst(pool, "1-2-2", 2, 'topdown', len=20, max_use=1)
x <- mst_obj(x, theta=-1, indices=1)
x <- mst_obj(x, theta=0, indices=2:3)
x <- mst_obj(x, theta=1, indices=4)
x <- mst_constraint(x, "content", 6, 6, level=1)
x <- mst_constraint(x, "content", 6, 6, level=2)
x <- mst_constraint(x, "content", 8, 8, level=3)
x <- mst_stage_length(x, 1:2, min=5)
x <- mst_assemble(x, timeout=3)

## ex. 1: administer the MST using fixed RDP for routing
x_sim <- mst_sim(x, .5, list(stage1=0, stage2=0))
plot(x_sim)

## ex. 2: administer the MST using the max. info. for routing
x_sim <- mst_sim(x, .5)
plot(x_sim, ylim=c(-5, 5))

## End(Not run)
```

`utils` *Utility functions*

Description

Utility functions

`rmse` computes the root mean squared error of two numeric vectors/matrices

`freq` computes the frequency and percentage of given values

`evaluate_3pl_estimation` evaluates estimation results against true values

`evaluate_gpcm_estimation` evaluates estimation results against true values

Usage

```
rmse(x, y)

freq(x, values = NULL)

hermite_gauss(num_quad = 20)

evaluate_3pl_estimation(data_tru, data_est)

evaluate_gpcm_estimation(data_tru, data_est)
```

Arguments

x	a vector/matrix of numeric values
y	a vector/matrix of numeric values
values	a vector of valid values, NULL for all values
num_quad	the number of quadrature points
data_tru	a list of true parameters
data_est	a list of estimated parameters

Examples

```
rmse(1 + rnorm(100), 2 + rnorm(100))
freq(sample(1:5, 100, replace=TRUE))
```

Index