Package ‘maat’

July 9, 2021

Type Package

Title Multiple Administrations Adaptive Testing

Version 1.0.2

Date 2021-07-08

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

Description Provides an extension of the shadow-test approach to computerized adaptive testing (CAT) implemented in the 'TestDesign' package for the assessment framework involving multiple tests administered periodically throughout the year. This framework is referred to as the Multiple Administrations Adaptive Testing (MAAT) and supports multiple item pools vertically scaled and multiple phases (stages) of CAT within each test. Between phases and tests, transitioning from one item pool (and associated constraints) to another is allowed as deemed necessary to enhance the quality of measurement.

URL https://choi-phd.github.io/maat/

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

License GPL (>= 2)

Depends R (>= 3.5.0)

biocViews

Imports TestDesign (>= 1.2.0), readxl, Rdpack, methods, MASS, diagram

Suggests testthat (>= 3.0.0), rmarkdown, knitr, kableExtra

RoxygenNote 7.1.1

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RdMacros Rdpack

Collate 'import.R' 'module_class.R' 'module_functions.R'
 'administered_functions.R' 'datasets.R' 'examinee_class.R'
 'examinee_updaters.R' 'extensions.R'
 'module_structure_operators.R' 'package.R' 'sim_functions.R'
 'plot_functions.R' 'print_functions.R' 'prior_functions.R'
 'routing_functions.R' 'show_functions.R'
Config/testthat/edition 3

NeedsCompilation no

Author  Seung W. Choi [aut, cre] (<https://orcid.org/0000-0003-4777-5420>),
        Sangdon Lim [aut] (<https://orcid.org/0000-0002-2988-014X>),
        Luping Niu [aut] (<https://orcid.org/0000-0003-3696-1180>),
        Sooyong Lee [aut] (<https://orcid.org/0000-0002-7964-4508>),
        M. Christina Schneider [ctb],
        Jay Lee [ctb],
        Garron Gianopulos [ctb]

Repository CRAN

Date/Publication 2021-07-09 08:30:02 UTC

R topics documented:

maat-package ............................................ 3
assessment_structure-class ................................. 3
boundGrade ............................................ 4
changeGrade ............................................ 5
changePhase ............................................ 6
changeTest ............................................ 6
createAssessmentStructure .................................. 7
createModule ............................................ 8
examinee-class ........................................... 8
excludeAdministeredItems .................................... 9
formatOutput ............................................ 10
getAdaptivityIndex ......................................... 12
getAdministeredItemsPerTest ................................ 12
getBias ................................................ 13
getItemExposureRate ........................................ 13
getItemNamesPerGrade ....................................... 14
getRelativeGrade .......................................... 14
getRMSE ............................................... 15
getSE ............................................... 15
loadModules ............................................ 16
maat .................................................. 17
module-class ............................................. 20
module_list_math ......................................... 20
output_maat-class ......................................... 20
plot ................................................ 21
print ............................................... 23
removeItemData ........................................... 23
show ................................................ 24
simExaminees ............................................. 24
simTheta ............................................... 25
updateGrade ............................................. 26
updateItemData ........................................... 28
maat-package

Description

Multiple Administrations Adaptive Testing

Details

maat package is based on the assessment framework involving multiple tests administered throughout the year using multiple item pools vertically scaled and multiple phases (stages) of computerized adaptive testing (CAT) within each test allowing for transitioning from one item pool (and associated constraints) to another between phases as determined necessary by a selected transition policy to enhance the quality of measurement.

The current version of maat supports three administrations (Fall, Winter, and Spring) with two phases within each administration (Phase 1 and Phase 2), for six modules in total administered over the course of a year.

Within each administration, students begin Phase 1 at the grade of record. One exception to this is that if a student’s final $\theta$ from the previous administration was above the ‘advanced achievement’ cut score of the grade of record, then the student begins Phase 1 of the following administration in an above-grade item pool. For example, if a Grade 3 student’s final $\theta$ from the Fall administration was $\theta = 1.1$ and the ‘advanced achievement’ cut score for Grade 3 was $\theta = 1.0$, then the student begins Phase 1 of the Winter administration in a Grade 4 item pool.

Within each administration, at the completion of Phase 1, business rules are used to determine whether a student is routed to an on-grade or off-grade item pool in Phase 2.

Detailed descriptions of the assessment design are available in the vignette.

assessment_structure-class

Class 'assessment_structure': assessment structure

Description

assessment_structure is an S4 class to represent an assessment structure.
**Slots**

- **n_test** a numeric, the number of test administrations.
- **n_phase** a numeric, the number of phases within each test.
- **route_limit_below** the number of grades to allow routing below, relative to the grade of record. If the grade of record is G4 and this is 1, then routing to G3 is allowed but not to G2.
- **route_limit_above** the number of grades to allow routing above, relative to the grade of record. If the grade of record is G4 and this is 2, then routing to G6 is allowed but not to G7.
- **test_routing_restrictions** R1: If grade is G-1 in the last phase of any administration, ignore achievement level and always change grade by +1. R2: If grade is G in the last phase of any administration: If achievement level is Beginning, do not decrease grade. R3: If grade is G+k in the last phase of Administration k: If achievement level is Advanced, do not increase grade.

---

**boundGrade**

*Bound grades within a specified range*

**Description**

*boundGrade* is a function for keeping the grade within a specified range. *boundGrade* checks the relative grade compared to the grade of record. If the current grade is outside the allowed bound, the grade that is within the bound in the same direction is returned.

**Usage**

```r
boundGrade(
  current_grade,
  grade_of_record,
  route_limit_below,
  route_limit_above
)
```

**Arguments**

- **current_grade** the current grade. This must be formatted as G?, where ? is a number.
- **grade_of_record** the grade of record. This must be formatted as G?, where ? is a number.
- **route_limit_below** the number of grades to allow routing below, relative to the grade of record. If the grade of record is G4 and this is 1, then routing to G3 is allowed but not to G2.
- **route_limit_above** the number of grades to allow routing above, relative to the grade of record. If the grade of record is G4 and this is 2, then routing to G6 is allowed but not to G7.
changeGrade

Value

the grade after the range limit is applied

Examples

boundGrade("G2", "G1", 0, 2) # G2
boundGrade("G3", "G1", 0, 2) # G3
boundGrade("G4", "G1", 0, 2) # G3
boundGrade("G5", "G1", 0, 2) # G3

changeGrade

Grade operator: add or subtract

Description

changeGrade is an operator for grade values.

Usage

changeGrade(grade, delta)

Arguments

grade a string containing the current grade in the form G?, where ? is a number.
delta a number containing the relative change in grade to apply. 0 retains the current grade as-is.

Value

a string containing the new grade.

Examples

changeGrade("G4", 0) ## G4
changeGrade("G4", 1) ## G5
changeGrade("G4", -1) ## G3
changeGrade("G10", 1) ## G11
### changePhase

**Phase operator: move to next phase**

**Description**

changePhase is an operator for phase values.

**Usage**

\[
\text{changePhase}(\text{phase}, \text{assessment\_structure})
\]

**Arguments**

- `phase` a string containing the current phase in the format P?, where ? is a number.
- `assessment\_structure` an assessment\_structure object.

**Value**

a string containing the new phase.

**Examples**

```
## assessment uses two phases
changePhase("P1", assessment\_structure_math) ## P2
changePhase("P2", assessment\_structure_math) ## P1
```

### changeTest

**Test operator: move to next phase**

**Description**

changeTest is an operator for test values.

**Usage**

\[
\text{changeTest}(\text{test}, \text{phase}, \text{assessment\_structure})
\]

**Arguments**

- `test` a string containing the current test in the format T?, where ? is a number.
- `phase` a string containing the current phase in the format P?, where ? is a number.
- `assessment\_structure` an assessment\_structure object.
createAssessmentStructure

Value

a string containing the new test.

Examples

```r
## assessment uses two phases
changeTest("T1", "P1", assessment_structure_math) ## T1
changeTest("T1", "P2", assessment_structure_math) ## T2
```

createAssessmentStructure

Create an assessment structure

Description

`createAssessmentStructure` is a function for creating an `assessment_structure` object that defines the structure of the assessment.

Usage

```r
createAssessmentStructure(
  n_test,
  n_phase,
  route_limit_below,
  route_limit_above,
  test_routing_restrictions = c("R1", "R2", "R3")
)
```

Arguments

- **n_test** a numeric, the number of test administrations.
- **n_phase** a numeric, the number of phases within each test.
- **route_limit_below** the number of grades to allow routing below, relative to the grade of record. If the grade of record is G4 and this is 1, then routing to G3 is allowed but not to G2.
- **route_limit_above** the number of grades to allow routing above, relative to the grade of record. If the grade of record is G4 and this is 2, then routing to G6 is allowed but not to G7.
- **test_routing_restrictions** the restrictions for between-test routing. (default = c("R1", "R2", "R3")

Value

an `assessment_structure` object.
Examples

```r
assessment_structure <- createAssessmentStructure(
  n_test = 3,
  n_phase = 2,
  route_limit_below = 1,
  route_limit_above = 2
)
```

---

**createModule**

*Create a single module*

**Description**

`createModule` is a function for creating a `module` object based on the item pool, attribute, and constraints.

**Usage**

```
createModule(constraints, item_pool, item_attrib, passage_attrib)
```

**Arguments**

- `constraints` constraints data. A `data.frame` or a csv file name to be used in `loadConstraints`.
- `item_pool` item pool data. A `data.frame` or a csv file name to be used in `loadItemPool`.
- `item_attrib` item attribute data. A `data.frame` or a csv file name to be used in `loadItemAttrib`.
- `passage_attrib` passage attribute data. A `data.frame` or a csv file name to be used in `loadStAttrib`.

**Value**

A `module` object.

---

**examinee-class**

*Class ‘examinee’: a single examinee*

**Description**

`examinee` is an S4 class to represent a single examinee.
**excludeAdministeredItems**

*Update a constraints object to exclude administered items*

---

**Slots**

- **examinee_id** the ID of examinee.
- **current_grade** the current grade the examinee is in.
- **current_phase** the current phase the examinee is in.
- **current_test** the current test the examinee is in.
- **current_module** the current module the examinee is in.
- **grade_log** grades that the examinee belonged at each module position.
- **phase_log** phases that the examinee belonged at each module position.
- **test_log** tests that the examinee belonged at each module position.
- **module_log** modules that the examinee belonged at each module position.
- **n_module** the number of modules the examinee received. This is the number of module positions.
- **true_theta** a vector containing the true theta (if simulated) of the examinee, for each module position.
- **initial_theta_in_module** a vector containing initial thetas used in each module.
- **prior_par_by_module** a list containing prior parameters used for each module.
- **estimated_theta_by_phase** a list containing estimated thetas and SEs using items in each phase.
- **estimated_theta_by_test** a list containing estimated thetas and SEs using combined items in each test.
- **estimated_theta_for_routing** a list containing estimated thetas and SEs that were used for routing.
- **alpha** the alpha value used to compute lower and upper bounds.
- **selection_theta** a list containing selection thetas in each module position.
- **interim_theta** a list containing interim thetas and SEs in each module position.
- **administered_items** a list containing administered items in each module position.
- **administered_stimuli** a list containing administered stimuli in each module position.
- **response** a list containing the examinee response in each module position.
- **item_data** a list containing item pool of administered items.
- **routing_based_on** a vector containing the routing was based on estimated_theta_by_phase or estimated_theta_by_test at each module position.

---

**Description**

The function `excludeAdministeredItems` produces a new `constraints` object that excludes administered items from being selected.
Usage

excludeAdministeredItems(constraints, administered_items)

Arguments

constraints a constraints object.
administered_items item names of previously administered items.

Value

a constraints object that also constrains the administered items to be excluded.

Examples

## Not run:
require(TestDesign)

cfg <- createShadowTestConfig(
  MIP = list(solver = "lpsymphony")
)
constraints <- constraints_reading
solution <- Shadow(cfg, constraints, true_theta = 0)
administered_items <- solution@output[[1]]@administered_item_index
administered_items <- solution@constraints@pool@id[administered_items]
administered_items

updated_constraints <- excludeAdministeredItems(constraints, administered_items)

solution <- Shadow(cfg, updated_constraints, true_theta = 0)
administered_items <- solution@output[[1]]@administered_item_index
administered_items <- solution@constraints@pool@id[administered_items]
administered_items ## entirely different from above

## End(Not run)

formatOutput

Format the output of maat

Description

formatOutput is a function for formatting the output examinee object of the function maat for analysis.

Usage

formatOutput(examinee_list, digits = 3)
Arguments

- **examinee_list**: the output from `maat`.
- **digits**: digits to round theta values. (default = 3)

Value

A data frame containing:

- **p_ID**: the person ID.
- **test_phase_ID**: the module position. If we have 3 tests with 2 phases in each test then the range of test_phase_ID is 1 to 6.
- **initial_grade**: the initial grade of the person.
- **final_grade**: the final grade of the person after completing all modules.
- **grade_ID**: the grade at the module position.
- **phase_ID**: the phase at the module position.
- **test_ID**: the test at the module position.
- **module_ID**: the module ID at the module position.
- **final_theta_est**: the grand final estimated $\theta$ after completing all tests.
- **final_SE_est**: the standard error of grand final estimated $\theta$ after completing all tests.
- **theta_by_phase**: the final estimated $\theta$ after completing each phase.
- **SE_by_phase**: the standard error of final estimated $\theta$ after completing each phase.
- **combined**: whether items were combined with the previous phase to obtain the theta estimate.
- **true_theta**: the true $\theta$ in each module position.
- **item_ID**: the item IDs of administered items.
- **ncat**: the number of categories of administered items.
- **IRT_model**: the IRT models of administered items.
- **item_par_1**: the first item parameter of each administered item (e.g., for 1PL, this is item difficulty)
- **item_par_2**: the second item parameter of each administered item (e.g., for 1PL, this is ‘NA’)
- **item_resp**: the item response on each administered item.
- **momentary_theta**: the momentary (interim) $\theta$ estimate obtained after each item administration in CAT engine.
- **momentary_SE**: the standard error of momentary (interim) $\theta$ estimate obtained after each item administration in CAT engine.
getAdaptivityIndex

*Calculate adaptivity indices from an examinee list object*

**Description**

`getAdaptivityIndex` is a function for calculating adaptivity indices from the output of `maat`.

**Usage**

`getAdaptivityIndex(x)`

**Arguments**

- `x` an `output_maat` object from `maat`.

**Value**

A data frame containing adaptivity indices by test and also for all tests combined.

getAdministeredItemsPerTest

*Get administered items per test*

**Description**

`getAdministeredItemsPerTest` is a function for extracting the administered items stored in the `examinee` objects.

**Usage**

`getAdministeredItemsPerTest(x)`

**Arguments**

- `x` an `output_maat` object from `maat`.

**Value**

A list containing administered items in each test and also for all tests combined.
**getBias**

*Calculate bias from an examinee list object*

**Description**

`getBias` is a function for calculating the bias of ability estimates of the simulation results.

**Usage**

```r
getBias(x)
```

**Arguments**

- `x` an `output_maat` object from `maat`.

**Value**

a list containing bias by test and also for all tests combined.

---

**getItemExposureRate**

*Get item exposure rates from an examinee list*

**Description**

`getItemExposureRate` is a function for building an item exposure rate table.

**Usage**

```r
g.GetItemExposureRate(x)
```

**Arguments**

- `x` an `output_maat` object from `maat`.

**Value**

the table of item exposure rate.
**getItemNamesPerGrade**  
*Get item names per grade*

**Description**

`getItemNamesPerGrade` is a function for extracting item names from a module list.

**Usage**

```plaintext
getItemNamesPerGrade(module_list)
```

**Arguments**

- `module_list`  
a module list from `loadModules`.

**Value**

item names per grade.

**Examples**

```plaintext
getItemNamesPerGrade(module_list_math)
```

---

**getRelativeGrade**  
*Grade operator: difference between two grades*

**Description**

`getRelativeGrade` is an operator for grade values.

**Usage**

```plaintext
getRelativeGrade(current_grade, initial_grade)
```

**Arguments**

- `current_grade`  
a string containing the current grade in the form G?, where ? is a number.
- `initial_grade`  
a string containing the initial grade in the form G?, where ? is a number.

**Value**

the grade difference of the current grade relative to the initial grade.
getRMSE

Examples
  getRelativeGrade("G4", "G3") ## 1
  getRelativeGrade("G5", "G3") ## 2
  getRelativeGrade("G2", "G3") ## -1

getRMSE

Calculate RMSE from an examinee list object

Description
  getRMSE is a function for calculating root mean square error (RMSE) for the simulation results.

Usage
  getRMSE(x)

Arguments
  x an output_maat object from maat.

Value
  a list containing RMSE by test and also for all tests combined.

getSE

Calculate standard error from an examinee list object

Description
  getSE is a function for calculating the standard error of the estimates.

Usage
  getSE(x)

Arguments
  x an output_maat object from maat.

Value
  a list containing SE by test and also for all tests combined.
loadModules | Load multiple modules

**Description**

`loadModules` is a function for creating multiple module objects from a specification sheet.

**Usage**

```r
loadModules(fn, base_path = NULL, assessment_structure, examinee_list)
```

**Arguments**

- `fn` | the name of a csv file containing module specifications.
- `base_path` | (optional) the base path to append before the file paths contained in module specs.
- `assessment_structure` | an assessment_structure object.
- `examinee_list` | an examinee list from `simExaminees`. Used to determine the range of required modules.

**Details**

The module specification file is expected to have the following columns:

- Grade a string containing the grade in the form `G?`, where `?` is a number.
- Phase a string containing the phase in the form `P?`, where `?` is a number.
- ItemPool the file path of a file that contains item pool data. This must be readable with `loadItemPool`.
- ItemAttrib the file path of a file that contains item attribute data. This must be readable with `loadItemAttrib`.
- PassageAttrib the file path of a file that contains passage attribute data. This must be readable with `loadStAttrib`.
- Constraints the file path of a file that contains constraints data. This must be readable with `loadConstraints`.

**Value**

a module list containing module objects. Each module can be accessed using `module_list[[grade]][[phase]]`. 
Examples

```r
assessment_structure <- createAssessmentStructure(
    n_test = 3,
    n_phase = 2,
    route_limit_below = 0,
    route_limit_above = 2
)
examinee_list <- simExaminees(
    N = 5,
    mean_v = c(0, 0, 0),
    sd_v = c(1, 1, 1),
    cor_v = diag(1, 3),
    assessment_structure = assessment_structure
)
```

```r
fn <- system.file("extdata", "module_definition_MATH_normal_N500.csv", package = "maat")
pkg_path <- system.file(package = "maat")
module_list <- loadModules(
    fn,
    base_path = pkg_path,
    assessment_structure = assessment_structure,
    examinee_list = examinee_list
)
```

---

**maat**

*Simulate multi-stage multi-administration adaptive test*

---

**Description**

*maat* is the main function for simulating a multi-stage multi-administration adaptive test.

**Usage**

```r
maat(
    examinee_list = examinee_list,
    assessment_structure,
    module_list,
    config,
    cut_scores,
    overlap_control_policy,
    transition_policy = "CI",
    combine_policy = "conditional",
    transition_CI_alpha = NULL,
    transition_percentile_lower = NULL,
    transition_percentile_upper = NULL,
    initial_theta_list = NULL,
    prior_mean_policy = "mean difficulté",
```
prior_mean_user = NULL,
prior_sd = 1,
verbose = TRUE
}

Arguments

examinee_list  an examinee list from simExaminees.
assessment_structure  a assessment_structure object.
module_list  a module list from loadModules.
config  a config_Shadow object.
cut_scores  a named list containing cut scores to be used in each grade. Each element must be named in the form G?, where ? is a number.
overlap_control_policy  overlap control is performed by excluding administered items from being administered again within the same examinee.
  • all performs overlap control at all module positions.
  • within_test performs overlap control only within each test.
  • none does not perform overlap control.
transition_policy  • CI uses the confidence interval to perform routing.
  • pool_difficulty_percentile uses item difficulty percentiles of all items in the item_pool argument to perform routing.
  • pool_difficulty_percentile_exclude_administered uses item difficulty percentiles of all items in the item_pool argument to perform routing, excluding all previous items administered to the examinee.
  • on_grade does not permit any transition.
  • (default = CI)
combine_policy  • This is only applied when module_position %% 2 == 0 (at Phase 2, which is the end of each test).
  • conditional uses the combined theta (using items from the previous module combined with the current module), if the examinee was in the same grade in Phases 1 and 2. If the examinee was in different grades in Phases 1 and 2, then the theta estimate from Phase 2 is used.
  • always uses the combined theta.
  • never uses the theta estimate from Phase 2.
  • (default = conditional)
transition_CI_alpha  the alpha level to use when transition_policy == "CI".
transition_percentile_lower  the percentile value (between 0 and 1) to use for the lower routing when transition_policy == "difficulty_percentile".
transition_percentile_upper  the percentile value (between 0 and 1) to use for the upper routing when transition_policy == "difficulty_percentile".
initial_theta_list
(optional) a list containing initial thetas to use in each module position.

prior_mean_policy
  • This is only effective at the beginning of each test. This determines what
    value is used as the prior mean.
  • mean_difficulty uses the mean item difficulty of the current item pool.
  • carryover uses the routing theta from the previous module. For Phase 1 of
    the first test, user supplied values are used if available. Otherwise, the
    mean item difficulty of the current item pool is used.
  • user uses user-supplied values in the prior_mean_user argument.
  • (default = mean_difficulty)

prior_mean_user
(optional) user-supplied values for the prior mean. Must be a single value, or a
vector for each grade.

prior_sd
user-supplied values for the prior standard deviation. This is only effective at
the beginning of each test. This is utilized regardless of prior_mean_policy.
Must be a single value, or a vector for each grade. (default = 1)

verbose
  if TRUE, print status messages. (default = TRUE)

Value

an output_maat object from the simulation.

Examples

library(TestDesign)
config <- createShadowTestConfig(
  final_theta = list(
    method = "MLE"
  )
)
examinee_list <- maat(
  examinee_list = examinee_list_math,
  assessment_structure = assessment_structure_math,
  module_list = module_list_math,
  overlap_control_policy = "all",
  transition_CI_alpha = 0.05,
  config = config,
  cut_scores = cut_scores_math
)
### module-class

**Class 'module': a module**

**Description**

*module* is an S4 class to represent a module.

**Slots**

- **module_id** the ID of the module.
- **constraints** a *constraints* object.

### module_list_math

**Example item pools**

**Description**

Example data for a 6-module assessment.

**Details**

- **assessment_structure_math** an *assessment_structure* object defining 3 tests with 2 phases in each test. Also defines routing limits as $G - 1$ and $G + 2$, where $G$ is the starting grade.
- **examinee_list_math** a list of *examinee* objects. The number of examinees is 10. This can be created using *simExaminees*.
- **module_list_math** a list of *module* objects. This can be created using *loadModules*.
- **cut_scores_math** a list of theta cut scores. This is used in the *cut_scores* argument of the *maat* function.

### output_maat-class

**Class 'output_maat': a simulation output**

**Description**

*output_maat* is an S4 class to represent a simulation output.
Slots

- `examinee_list`: a list of `examinee` objects.
- `assessment_structure`: an `assessment_structure` object.
- `module_list`: a module list from `loadModules`.
- `config`: the `config_Shadow` object used in the simulation.
- `cut_scores`: the cut scores used in the simulation.
- `overlap_control_policy`: the policy used in the simulation.
- `transition_policy`: the policy used in the simulation.
- `combine_policy`: the policy used in the simulation.
- `transition_CI_alpha`: the transition parameter used in the simulation.
- `transition_percentile_lower`: the transition parameter used in the simulation.
- `transition_percentile_upper`: the transition parameter used in the simulation.
- `initial_theta_list`: the starting theta values used in the simulation.
- `prior_mean_policy`: the policy used in the simulation.
- `prior_mean_user`: the prior parameters used in the simulation.
- `prior_sd`: the prior parameters used in the simulation.

---

**plot**  
*Extension of plot()*

---

**Description**

Extension of plot()

**Usage**

```r
## S4 method for signature 'output_maat'
plot(
x,  
y,  
type,  
examinee_id,  
cut_scores = NULL,  
theta_range = c(-4, 4),  
main = NULL,  
box_color = "PaleTurquoise"
)
```
Arguments

x x

y y

type the type of plot. route plots the number of examinees routed to each path across the course of entire assessment. correlation produces a scatterplot of thetas across administrations. audit plots interim thetas over modules for a single examinee.

examinee_id the examinee ID to plot.

cut_scores (optional) a named list containing cut scores for each grade.

theta_range the theta range to use in scatter plots when x is an examinee list.

main the figure title to use in scatter plots when x is an examinee list.

box_color the cell color to use when type is route. (default = PaleTurquoise)

Value

the route plot.

Examples

library(TestDesign)
config <- createShadowTestConfig(
  final_theta = list(
    method = "MLE"
  )
)
examinee_list <- maat(
  examinee_list = examinee_list_math,
  assessment_structure = assessment_structure_math,
  module_list = module_list_math,
  overlap_control_policy = "all",
  transition_CI_alpha = 0.05,
  config = config,
  cut_scores = cut_scores_math
)

plot(examinee_list, type = "route")
plot(examinee_list, type = "correlation")
plot(examinee_list, type = "audit", examinee_id = 1)
### print

#### Extension of print()

**Description**

Extension of print()

**Usage**

```r
## S4 method for signature 'module'
print(x)
```

**Arguments**

`x` an object to display the content.

---

### removeItemData

#### Remove item data from examinee list

**Description**

`removeItemData` is a function to remove the item data from the `examinee` objects for the reduction of file size.

**Usage**

`removeItemData(examinee_list)`

**Arguments**

`examinee_list` a list containing `examinee` objects.

**Value**

a list containing `examinee` objects, with item_data data stripped for compact storage.
**Description**

Extension of show()

**Usage**

```r
## S4 method for signature 'module'
show(object)
```

**Arguments**

- `object` an object to display the content.

---

**Description**

`simExaminees` is a function for generating a list of `examinee` objects.

**Usage**

```r
simExaminees(
  N,
  mean_v,
  sd_v,
  cor_v,
  assessment_structure,
  initial_grade = "G4",
  initial_phase = "P1",
  initial_test = "T1"
)
```

**Arguments**

- `N` the number of examinees.
- `mean_v` a vector containing the mean of each dimension.
- `sd_v` a vector containing the standard deviation of each dimension.
- `cor_v` a correlation matrix.
- `assessment_structure` an `assessment_structure` object. This can be created using `createAssessmentStructure`. 
initial_grade: the initial grade for all examinees. The grade must exist in module_list. (default = G4)

initial_phase: the initial phase for all examinees. The phase must exist in module_list. (default = P1)

initial_test: the initial test for all examinees. (default = T1)

Details

Each dimension of mean_v, sd_v, cor_v represents a test level. For example in a three-test structure (see the assessment_structure_math example data), these arguments must have three dimensions.

Value

A list of examinee objects.

Examples

assessment_structure <- createAssessmentStructure(
  n_test = 3,
  n_phase = 2,
  route_limit_below = 1,
  route_limit_above = 2
)

examinee_list <- simExaminees(
  N = 100,
  mean_v = c(0, 0, 0),
  sd_v = c(1, 1, 1),
  cor_v = diag(3),
  assessment_structure = assessment_structure
)

---

**simTheta**

*Simulate theta values*

Description

**simTheta** is a function for generating a theta matrix based on the given sample size, mean, standard deviation, and correlation matrix.

Usage

simTheta(N, mean_v, sd_v, cor_v)
updateGrade

Arguments

N       the number of examinees.
mean_v  a vector containing the mean of each dimension.
sd_v    a vector containing the standard deviation of each dimension.
cor_v   a correlation matrix.

Details

simTheta calls mvrnorm internally.

Value

a theta matrix.

Examples

```r
o <- simTheta(
  N = 100,
  mean_v = c(0, 0, 0),
  sd_v = c(1, 1, 1),
  cor_v = diag(1, 3)
)
```

updateGrade (Update the grade slot of an examinee object)

Description

updateGrade is a function for determining the grade an examinee is routed to.

Usage

```r
updateGrade(
  examinee_object,
  assessment_structure,
  module_position,
  cut_scores,
  transition_policy = "CI",
  transition_CI_alpha,
  transition_percentile_lower,
  transition_percentile_upper,
  item_pool
)
```
Arguments

examinee_object
an `examinee` object.

assessment_structure
an `assessment_structure` object.

module_position
the current module position, ranging from 1 to 6.

cut_scores
a named list containing cut scores to be used in each grade. Each element must be named in the form G?, where ? is a number.

transition_policy

• CI uses the confidence interval to perform routing.
• `pool_difficulty_percentile` uses item difficulty percentiles of all items in the `item_pool` argument to perform routing.
• `pool_difficulty_percentile_exclude_administered` uses item difficulty percentiles of all items in the `item_pool` argument to perform routing, excluding all previous items administered to the examinee.
• on_grade does not permit any transition.
• (default = CI)

cut_scores
the `item_pool` object to determine difficulty range in percentile-based transition policies.

Details

Currently the routing rules are hard-coded in the function. See the vignette for a description of routing rules.

Value

an `examinee` object with its current_grade slot updated.
updateItemData  

*Update the item data slot of an examinee object*

**Description**

`updateItemData` is a function for updating `examinee` objects after completing a module.

**Usage**

`updateItemData(examinee_object, module_position, solution)`

**Arguments**

- `examinee_object`  
  an `examinee` object.
- `module_position`  
  the current module position.
- `solution`  
  an `output_Shadow_all` object.

**Details**

`updateItemData` updates the `item_data` slot with an `item_pool` object that contains administered items in the module.

**Value**

an `examinee` object with its `item_data` slot updated.

updateLog  

*Update the routing log of an examinee object*

**Description**

`updateLog` is a function for updating `examinee` objects after completing a module. `updateLog` updates logs with grades, phases, tests and modules.

**Usage**

`updateLog(examinee_object, current_module_position)`

**Arguments**

- `examinee_object`  
  an `examinee` object.
- `current_module_position`  
  the current module position.
**updateModule**

*Description*

`updateModule` is a function for updating `examinee` objects after completing a module. `updateModule` assigns an `module` object from the supplied list to match the grade and the phase the `examinee` is in.

*Usage*

`updateModule(examinee_object, module_list)`

*Arguments*

- `examinee_object` an `examinee` object.
- `module_list` a module list from `loadModules`.

*Value*

an `examinee` object with its current_module slot updated.

**updatePhase**

*Description*

`updatePhase` is a function for updating `examinee` objects after completing a module. `updatePhase` updates the phase by calling `changePhase`.

*Usage*

`updatePhase(examinee_object, assessment_structure)`

*Arguments*

- `examinee_object` an `examinee` object.
- `assessment_structure` an `assessment_structure` object.
updateTest

Update the current test of an examinee object

Description

tupdateTest is the function for updating the new test ID in an examinee object.

Usage

updateTest(examinee_object, assessment_structure)

Arguments

examinee_object
an examinee object.

assessment_structure
an assessment_structure object.

Value

an examinee object with its current_test slot updated.

Examples

## assessment uses two phases

examinee <- examinee_list_math[[1]]
examinee@current_phase ## P1

examinee <- updatePhase(examinee, assessment_structure_math)
examinee@current_phase ## P2

examinee <- updatePhase(examinee, assessment_structure_math)
examinee@current_phase ## P1

examinee <- updateTest(examinee, assessment_structure_math)
examinee <- updatePhase(examinee, assessment_structure_math)
updateThetaForRouting

examinee@current_test  ## T1
examinee@current_phase  ## P2

examinee <- updateTest(examinee, assessment_structure_math)
examinee <- updatePhase(examinee, assessment_structure_math)
examinee@current_test  ## T2
examinee@current_phase  ## P1

updateThetaForRouting  Update the theta used for routing of an examinee object

Description

updateThetaForRouting is a function for updating examinee objects after completing a module. updateThetaForRouting determines what type of theta estimate is used to perform routing.

Usage

updateThetaForRouting(examinee_object, current_module_position, combine_policy)

Arguments

examinee_object
   an examinee object.

current_module_position
   the current module position.

combine_policy
   • This is only applied when module_position %% 2 == 0 (at Phase 2, which is the end of each test).
   • conditional uses the combined theta (using items from the previous module combined with the current module), if the examinee was in the same grade in Phases 1 and 2. If the examinee was in different grades in Phases 1 and 2, then the theta estimate from Phase 2 is used.
   • always uses the combined theta.
   • never uses the theta estimate from Phase 2.
   • (default = conditional)

Value

an examinee object with its estimated_theta_for_routing slot updated.
updateThetaUsingCombined

*Update theta estimates using combined responses from a test*

**Description**

`updateThetaUsingCombined` is a function for updating *examinee* objects after completing a module. `updateThetaUsingCombined` adds final theta estimates using all administered items in the test. A test may consist of multiple phases.

**Usage**

`updateThetaUsingCombined(examinee_object, current_module_position, config)`

**Arguments**

- `examinee_object`  
  an *examinee* object.
- `current_module_position`  
  the current module position.
- `config`  
  a `config_Shadow` object. The config for obtaining final estimates is used.

**Value**

an *examinee* object with its `estimated_theta_by_test` slot updated.
Index

* datasets
  - module_list_math, 20

* package
  - maat-package, 3

assessment_structure, 3, 6, 7, 16, 18, 20, 21, 24, 27, 29, 30
assessment_structure-class, 3
assessment_structure_math  
  (module_list_math), 20

boundGrade, 4, 4
changeGrade, 5, 5
changePhase, 6, 6, 29
changeTest, 6, 6
config_Shadow, 18, 21, 32
constraints, 9, 10, 20
createAssessmentStructure, 7, 7, 24
createModule, 8, 8
cut_scores_math (module_list_math), 20
data.frame, 8

examinee, 8, 10, 12, 20, 21, 23–25, 27–32
examinee-class, 8
examinee_list_math (module_list_math), 20
excludeAdministeredItems, 9, 9

formatOutput, 10, 10

getAdaptivityIndex, 12, 12
getAdministeredItemsPerTest, 12, 12
getBias, 13, 13
getItemExposureRate, 13, 13
g getItemNamesPerGrade, 14, 14
getRelativeGrade, 14, 14
getRMSE, 15, 15
getSE, 15, 15

item_pool, 9, 27, 28

loadConstraints, 8, 16
loadItemAttrib, 8, 16
loadItemPool, 8, 16
loadModules, 14, 16, 18, 20, 21, 29
loadStAttrib, 8, 16
maat, 10–13, 15, 17, 17, 20
maat-package, 3
module, 8, 16, 20, 29
module-class, 20
module_list_math, 20
mvrnorm, 26

output_maat, 12, 13, 15, 19, 20
output_maat-class, 20
output_Shadow_all, 28

plot, 21
plot, output_maat-method (plot), 21
print, 23
print, module-method (print), 23

removeItemData, 23, 23

show, 24
show, module-method (show), 24
simExaminees, 16, 18, 20, 24, 24
simTheta, 25, 25, 26

updateGrade, 26, 26
updateItemData, 28, 28
updateLog, 28, 28
updateModule, 29, 29
updatePhase, 29, 29
updateTest, 30, 30
updateThetaForRouting, 31, 31
updateThetaUsingCombined, 32, 32