Package: mirtCAT (via r-universe)

October 14, 2026

Version 1.14

Type Package

Title Computerized Adaptive Testing with Multidimensional Item Response Theory

Description Provides tools to generate HTML interfaces for adaptive and non-adaptive tests using the shiny package (Chalmers (2016) <doi:10.18637/jss.v071.i05>). Suitable for applying unidimensional and multidimensional computerized adaptive tests (CAT) using item response theory methodology and for creating simple questionnaires forms to collect response data directly in R. Additionally, optimal test designs (e.g., ``shadow testing'') are supported for tests that contain a large number of item selection constraints. Finally, package contains tools useful for performing Monte Carlo simulations for studying test item banks.

Depends mirt (>= 1.37), shiny (>= 1.0.1)

Imports lattice, stats, Rcpp, methods, markdown, pbapply, lpSolve

Suggests shinythemes, parallel, knitr

Encoding UTF-8

ByteCompile yes

LazyLoad yes

VignetteBuilder knitr

LinkingTo Rcpp, RcppArmadillo

License GPL (>= 3)

Maintainer Phil Chalmers <rphilip.chalmers@gmail.com>

URL https://github.com/philchalmers/mirtCAT, https://github.com/philchalmers/mirtCAT/wiki, https://groups.google.com/forum/#!forum/mirt-package

BugReports https://github.com/philchalmers/mirtCAT/issues?state=open

RoxygenNote 7.3.2

NeedsCompilation yes

Author Phil Chalmers [aut, cre]
(<https://orcid.org/0000-0001-5332-2810>), Magnus Nordmo [ctb]
(<https://orcid.org/0000-0002-1977-1038>)

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

Date/Publication 2024-07-13 02:43:43 UTC

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

RemoteRef HEAD

RemoteSha 4d529fe92e732b9c0e8b85bbc2222f2cbad0320d

Contents

mirtCAT-package 2
computeCriteria 3
createSessionName 5
createShinyGUI 5
extract.mirtCAT 6
findNextItem 10
generate.mirt_object 14
generate_pattern 16
get_mirtCAT_env 17
getPerson 18
mirtCAT 19
mirtCAT_preamble 35
updateDesign 36

Index **38**

`mirtCAT-package` *Computerized Adaptive Testing with Multidimensional Item Response Theory*

Description

Computerized Adaptive Testing with Multidimensional Item Response Theory

Details

Provides tools to generate an HTML interface for creating adaptive and non-adaptive educational and psychological tests using the shiny package. Suitable for applying unidimensional and multidimensional computerized adaptive tests using item response theory methodology and for creating simple questionnaires forms to collect response data directly in R.

Users interested in the most recent version of this package can visit https://github.com/philchalmers/mirtCAT and follow the instructions for installing the package from source (additional details

about installing from Github can be found at https://github.com/philchalmers/mirt). Questions regarding the package can be sent to the mirt-package Google Group, located at https://groups.google.com/forum/#!forum/mirt-package.

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

References

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

---

computeCriteria *Compute the values given the criteria and internal objects*

Description

A function that returns a named vector of evaluated criteria for each respective item in the test bank. The names are associated with the item number in the bank. Note that criteria values are returned such that the maximum value always represents the most optimal item (e.g., maximum information). In cases where the minimum value is typically selected (e.g., minimum variance) all values are multiplied by -1 to turn it into a maximization problem.

Usage

```
computeCriteria(
  x,
  criteria,
  person = NULL,
  test = NULL,
  design = NULL,
  subset = NULL,
  info_mats = FALSE
)
```

Arguments

x	an object of class 'mirtCAT_design' returned from the mirtCAT function when passing design_elements = TRUE
criteria	item selection criteria (see mirtCAT's criteria input)
person	(required when x is missing) internal person object. To be used when customNextItem function has been defined

test	(required when x is missing) internal test object. To be used when customNextItem function has been defined
design	(required when x is missing) internal design object. To be used when customNextItem function has been defined
subset	an integer vector indicating which items should be included in the optimal search; the default NULL includes all possible items. To allow only the first 10 items to be selected from this can be modified to subset = 1:10. This is useful when administering a multi-unidimensional CAT session where unidimensional blocks should be clustered together for smoother presentation. Useful when using the customNextItem function in mirtCAT
info_mats	logical; if more than one trait is present in the test, should the respective information matricies be returned instead of the scalar summary statistics (e.g., D-rule). When TRUE will return a list of matricies associated with each respective item

Value

a vector of criteria values for each respective item

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

References

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

See Also

mirtCAT, updateDesign, extract.mirtCAT, findNextItem

Examples

```
## Not run:
# test defined in mirtCAT help file, first example
CATdesign <- mirtCAT(df, mod, design_elements = TRUE)

computeCriteria(CATdesign, criteria = 'MI')
computeCriteria(CATdesign, criteria = 'MEI')


## End(Not run)
```

createSessionName *Create a unique GUI session name from a string of characters*

Description

This is used in mirtCAT to create a random session name so that shiny knows which environment to select objects from when multiple CAT sessions have been initialized.

Usage

```
createSessionName(n = 30, datetime = TRUE)
```

Arguments

n	number of upper/lower characters to sample
datetime	logical; include the current date/time the function was called in the string as well? This further helps with the uniqueness of the generated string

Value

a list containing the internal environmental components for mirtCAT

createShinyGUI *Function returning an object used by shiny*

Description

This function returns the GUI setup results by calling shinyApp. Primarily, this is only useful when hosting the application publicly, such as through https://www.shinyapps.io/. The function mirtCAT_preamble must be run *before* this function is called. The object is executed by calling runApp.

Usage

```
createShinyGUI(ui = NULL, host_server = TRUE)
```

Arguments

ui	a shiny UI function used to define the interface. If NULL, the default one will be used. See mirtCAT:::default_UI for the internal code
host_server	logical; is createShinyGUI() being used on a remote server or executed locally? When TRUE any calls to stopApp are suppressed to allow for multiple sessions to be executed. Note that FALSE gives the same behaviour as the GUI in mirtCAT

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

References

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

See Also

mirtCAT, mirtCAT_preamble, getPerson

Examples

```
## Not run:

mirtCAT_preamble(df=df)
runApp(createShinyGUI(host_server = FALSE), port = 8000) # run locally

person <- getPerson()
summary(person)

runApp(createShinyGUI(), port = 8000) # for remote server hosting


## End(Not run)
```

---

extract.mirtCAT *Extract elements from the internal person, test, and design objects*

Description

This function extracts elements, as well as builds a few convenient elements, from the three internal person, design, or test objects that are accessible through a customNextItem function definition (see mirtCAT for details).

Usage

```
extract.mirtCAT(x, what)
```

Arguments

x	either the person, design, or test object defined through a customNextItem definition
what	a character vector extracting the desired element (see the Details section)

Details

Depending on which object is supplied, the following elements can be extracted.

The 'person' argument

`ID` a scalar value indicating the ID of the participant (generally only needed in Monte Carlo simulations)

`responses` an integer vector indicating how items that have been responded to. Each element pertains to the associated item location (e.g., `responses[100]` is associated with the 100th item), and is `NA` if the item has not been responded to

`raw_responses` of the same form as `responses`, pertaining to the observed responses in a character vector

`items_in_bank` an integer vector indicating items which have not been administered yet and are also valid candidates for administration

`items_answered` an integer vector indicating the order in which items have been responded to

`thetas` the current ability/latent trait estimates given the previously administered items

`thetas_SE` the current ability/latent trait standard error estimates given the previously administered items

`thetas_history` history of the ability/latent trait estimates

`thetas_SE_history` history of the latent trait standard error estimates

`item_time` of the same form as `items_answered`, pertaining to the amount of time it took the participant to response to the item

`demographics` a data.frame containing the (optional) prior survey information from the GUI interface

`clientData` a list of useful information from shiny's `session$clientData`

The 'design' argument

`items_not_scored` an integer vector indicating items which should be included but not scored in the test (these are experimental items)

`min_items` minimum number of items to administer

`max_items` maximum number of items to administer

`max_time` maximum amount of time alloted to the GUI

`met_SEM` logical vector indicating whether the SEM criteria has been met

`met_delta_thetas` logical vector indicating whether the delta_thetas criteria has been met

`met_classify` logical vector indicating whether the classify criteria has been met

`exposure` exposure control elements of the same form as `responses`

`content` content constraint information

`content_prop` content proportions

`test_properties` user-defined `data.frame` of test-based properties

`person_properties` user-defined `data.frame` of person-based properties

The 'test' argument

`mo` extract the defined model from the `mirt` package. Afterward, users can use the `extract.mirt` function to pull out a large number of internal elements for easy use

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

References

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

See Also

`mirt`, `mirtCAT`, `extract.mirt`, `findNextItem`

Examples

```
## Not run:
 #example test
set.seed(1234)
nitems <- 25
itemnames <- paste0('Item.', 1:nitems)
a <- matrix(rlnorm(nitems, .2, .3))
d <- matrix(rnorm(nitems))
dat <- simdata(a, d, 500, itemtype = 'dich')
colnames(dat) <- itemnames
mod <- mirt(dat, 1, verbose = FALSE, TOL = .01)

# simple math items
questions <- answers <- character(nitems)
choices <- matrix(NA, nitems, 5)
spacing <- floor(d - min(d)) + 1 #easier items have more variation in the options

for(i in 1:nitems){
  n1 <- sample(1:50, 1)
  n2 <- sample(51:100, 1)
  ans <- n1 + n2
  questions[i] <- paste0(n1, ' + ', n2, ' = ?')
  answers[i] <- as.character(ans)
  ch <- ans + sample(c(-5:-1, 1:5) * spacing[i,], 5)
  ch[sample(1:5, 1)] <- ans
  choices[i, ] <- as.character(ch)
}

df <- data.frame(Question=questions, Option=choices,
  Type = 'radio', stringsAsFactors = FALSE)
```

```
df$Answer <- answers

pat <- generate_pattern(mod, Theta = 0, df)

#------------------------------------------------
# administer items in sequence
customNextItem <- function(person, design, test){
   # browser()
   items_left_2_choose_from <- extract.mirtCAT(person, 'items_in_bank')
   min(items_left_2_choose_from)
}

res <- mirtCAT(df, local_pattern=pat,
  design = list(customNextItem=customNextItem))
summary(res)

#------------------------------------------------
# administer items in order, but stop after 10 items
customNextItem <- function(person, design, test){
   items_left_2_choose_from <- extract.mirtCAT(person, 'items_in_bank')
   items_answered <- extract.mirtCAT(person, 'items_answered')
   total <- sum(!is.na(items_answered))
   ret <- if(total < 10) min(items_left_2_choose_from)
     else return(NA)
   ret
}

res <- mirtCAT(df, local_pattern=pat,
  design = list(customNextItem=customNextItem))
summary(res)

#------------------------------------------------
# using findNextItem() and stopping after 10 items

customNextItem <- function(person, design, test){
   items_answered <- extract.mirtCAT(person, 'items_answered')
   total <- sum(!is.na(items_answered))
   ret <- NA
   if(total < 10)
     ret <- findNextItem(person=person, test=test, design=design, criteria = 'MI')
   ret
}

res <- mirtCAT(df, mod, local_pattern=pat, start_item = 'MI',
  design = list(customNextItem=customNextItem))
summary(res)

# equivalent to the following
res2 <- mirtCAT(df, mod, local_pattern=pat, start_item = 'MI',
  criteria = 'MI', design = list(max_items = 10))
summary(res2)
```

```
## End(Not run)
```

`findNextItem` *Find next CAT item*

Description

A function that returns the next item in the computerized adaptive, optimal assembly, or shadow test. For direction manipulation of the internal objects this function should be used in conjunction with the `updateDesign` and `customNextItem`. Finally, the raw input forms can be used when a `customNextItem` function has been defined in `mirtCAT`.

Usage

```
findNextItem(
  x,
  person = NULL,
  test = NULL,
  design = NULL,
  criteria = NULL,
  objective = NULL,
  subset = NULL,
  all_index = FALSE,
  ...
)
```

Arguments

`x`	an object of class 'mirtCAT_design' returned from the `mirtCAT` function when passing `design_elements = TRUE`
`person`	(required when `x` is missing) internal person object. To be used when `customNextItem` function has been defined
`test`	(required when `x` is missing) internal test object. To be used when `customNextItem` function has been defined
`design`	(required when `x` is missing) internal design object. To be used when `customNextItem` function has been defined
`criteria`	item selection criteria (see `mirtCAT`'s `criteria` input). If not specified the value from `extract.mirtCAT(design, 'criteria')` will be used
`objective`	a vector of values used as the optimization criteria to be passed to `lp(objective.in)`. This is typically the vector of criteria values returned from `computeCriteria`, however supplying other criteria are possible (e.g., to minimize the number of items administered simply pass a vector of -1's)
`subset`	an integer vector indicating which items should be included in the optimal search; the default `NULL` includes all possible items. To allow only the first 10 items to be selected from this can be modified to `subset = 1:10`. This is useful when administering a multi-unidimensional CAT session where unidimensional blocks

should be clustered together for smoother presentation. Useful when using the customNextItem function in mirtCAT

all_index logical; return all items instead of just the most optimal? When TRUE a vector of items is returned instead of the most optimal, where the items are sorted according to how well they fit the criteria (e.g., the first element is the most optimal, followed by the second most optimal, and so on). Note that this does not work for some selection criteria (e.g., 'seq' or 'random')

... additional arguments to be passed to lp

Details

When a numeric objective is supplied the next item in the computerized adaptive test is found via an integer solver through searching for a maximum. The raw input forms can be used when a customNextItem function has been defined in mirtCAT, and requires the definition of a constr_fun (see the associated element in mirtCAT for details, as well as the examples below). Can be used to for 'Optimal Test Assembly', as well as 'Shadow Testing' designs (van der Linden, 2005), by using the lp function. When objective is not supplied the result follows the typical maximum criteria of more standard adaptive tests.

Value

typically returns an integer value indicating the index of the next item to be selected or a value of NA to indicate that the test should be terminated. However, see the arguments for further returned object descriptions

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

References

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

van der Linden, W. J. (2005). Linear models for optimal test design. Springer.

See Also

mirtCAT, updateDesign, extract.mirtCAT

Examples

```
## Not run:

# test defined in mirtCAT help file, first example
# equivalent to criteria = 'MI'
customNextItem <- function(design, person, test){
```

```
   item <- findNextItem(person=person, design=design, test=test,
                        criteria = 'MI')
   item
 }

set.seed(1)
nitems <- 100
itemnames <- paste0('Item.', 1:nitems)
a <- matrix(rlnorm(nitems, .2, .3))
d <- matrix(rnorm(nitems))
dat <- simdata(a, d, 500, itemtype = 'dich')
colnames(dat) <- itemnames
mod <- mirt(dat, 1, verbose = FALSE)

# simple math items
questions <- answers <- character(nitems)
choices <- matrix(NA, nitems, 5)
spacing <- floor(d - min(d)) + 1 #easier items have more variation in the options

for(i in 1:nitems){
 n1 <- sample(1:50, 1)
 n2 <- sample(51:100, 1)
 ans <- n1 + n2
 questions[i] <- paste0(n1, ' + ', n2, ' = ?')
 answers[i] <- as.character(ans)
 ch <- ans + sample(c(-5:-1, 1:5) * spacing[i,], 5)
 ch[sample(1:5, 1)] <- ans
 choices[i, ] <- as.character(ch)
}

df <- data.frame(Question=questions, Option=choices,
              Type = 'radio', stringsAsFactors = FALSE)

response <- generate_pattern(mod, 1)
result <- mirtCAT(mo=mod, local_pattern = response,
                  design = list(customNextItem=customNextItem))

------------------------------------------------------------
# direct manipulation of internal objects
CATdesign <- mirtCAT(df=df, mo=mod, criteria = 'MI', design_elements = TRUE)

# returns number 1 in this case, since that's the starting item
findNextItem(CATdesign)

# determine next item if item 1 and item 10 were answered correctly
CATdesign <- updateDesign(CATdesign, new_item = 1, new_response = 1)
extract.mirtCAT(CATdesign$person, 'thetas') # updated thetas
CATdesign <- updateDesign(CATdesign, new_item = 10, new_response = 1)
extract.mirtCAT(CATdesign$person, 'thetas') # updated thetas again
findNextItem(CATdesign)
findNextItem(CATdesign, all_index = TRUE) # all items rank in terms of most optimal

#------------------------------------------------------------
```

```
## Integer programming example (e.g., shadow testing)

# find maximum information subject to constraints
#  sum(xi) <= 5              ### 5 or fewer items
#  x1 + x2 <= 1              ### items 1 and 2 can't be together
#  x4 == 0                   ### item 4 not included
#  x5 + x6 == 1              ### item 5 or 6 must be included, but not both

# constraint function
constr_fun <- function(design, person, test){

  # left hand side constrains
  #    - 1 row per constraint, and ncol must equal number of items
  mo <- extract.mirtCAT(test, 'mo')
  nitems <- extract.mirt(mo, 'nitems')
  lhs <- matrix(0, 4, nitems)
  lhs[1,] <- 1
  lhs[2,c(1,2)] <- 1
  lhs[3, 4] <- 1
  lhs[4, c(5,6)] <- 1

  # relationship direction
  dirs <- c("<=", "<=", '==', '==')

  #right hand side
  rhs <- c(5, 1, 0, 1)

  #all together
  constraints <- data.frame(lhs, dirs, rhs)
  constraints
}

CATdesign <- mirtCAT(df=df, mo=mod, design_elements = TRUE,
                     design = list(constr_fun=constr_fun))

# MI criteria value associated with each respective item
objective <- computeCriteria(CATdesign, criteria = 'MI')

# most optimal item, given constraints
findNextItem(CATdesign, objective=objective)

# all the items which solve the problem
findNextItem(CATdesign, objective=objective, all_index = TRUE)

## within a customNextItem() definition the above code would look like
# customNextItem <- function(design, person, test){
#   objective <- computeCriteria(person=person, design=design, test=test,
#                                criteria = 'MI')
#   item <- findNextItem(person=person, design=design, test=test,
#                        objective=objective)
#   item
# }
```

```
## End(Not run)
```

`generate.mirt_object` *Generate a mirt object from population parameters*

Description

This function generates a `mirt` object from known population parameters, which is then passed to `mirtCAT` for running CAT applications.

Usage

```
generate.mirt_object(
  parameters,
  itemtype,
  latent_means = NULL,
  latent_covariance = NULL,
  key = NULL,
  min_category = rep(0L, length(itemtype))
)
```

Arguments

`parameters`	a matrix or data.frame of parameters corresponding to the model definitions listed in `mirt`. Each row represents a unique item, while the column names correspond to the respective parameter names. If a parameter is not relevant for a particular item/row then use `NA`'s as placeholders
`itemtype`	a character vector indicating the type of item with which the parameters refer. See the `itemtype` argument in `mirt`. Note that this input is only used to determine the relevant item class for the rows in `parameters`, therefore many inputs are interchangeable (e.g., '2PL' generates the same internal model object as '3PL'). If only a single value is provided then all items types will be assumed identical
`latent_means`	(optional) a numeric vector used to define the population latent mean structure. By default the mean structure is centered at a 0 vector
`latent_covariance`	(optional) a matrix used to define the population variance-covariance structure between the latent traits. By default the relationship is assumed to be orthogonal standard normal (i.e., an identity matrix)
`key`	scoring key required for nested-logit models. See `mirt` for details
`min_category`	the value representing the lowest category index. By default this is 0, therefore the response suitable for the first category is 0, second is 1, and so on up to `K - 1`

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

References

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

See Also

mirt, mirtCAT, generate_pattern

Examples

```
## Not run:

### build a unidimensional test with all 3PL items

nitems <- 50
a1 <- rlnorm(nitems, .2,.2)
d <- rnorm(nitems)
g <- rbeta(nitems, 20, 80)

pars <- data.frame(a1=a1, d=d, g=g)
head(pars)

obj <- generate.mirt_object(pars, '3PL')
coef(obj, simplify = TRUE)
plot(obj, type = 'trace')

### build a two-dimensional test
## all graded items with 5 response categories

nitems <- 30
as <- matrix(rlnorm(nitems*2, .2, .2), nitems)
diffs <- t(apply(matrix(runif(nitems*4, .3, 1), nitems), 1, cumsum))
diffs <- -(diffs - rowMeans(diffs))
ds <- diffs + rnorm(nitems)
pars2 <- data.frame(as, ds)
colnames(pars2) <- c('a1', 'a2', paste0('d', 1:4))
head(pars2)

obj <- generate.mirt_object(pars2, 'graded')
coef(obj, simplify = TRUE)

### unidimensional mixed-item test

library(plyr)
```

```
pars3 <- rbind.fill(pars, pars2) #notice the NA's where parameters do not exist
obj <- generate.mirt_object(pars3, itemtype = c(rep('2PL', 50), rep('graded', 30)))
coef(obj)
itemplot(obj, 51)
itemplot(obj, 1, drop.zeros=TRUE)


## End(Not run)
```

`generate_pattern` *Generate a CAT patterns*

Description

Generate a CAT pattern given various inputs. Returns a character vector or numeric matrix (depending on whether a `df` input was supplied) with columns equal to the test size and rows equal to the number of rows in `Theta`. For simulation studies, supplying a `Theta` input with more than 1 row will generate a matrix of responses for running independent CAT session when passed to `mirtCAT(..., local_pattern)`. When the returned object is an integer vector then the `Theta` values will be stored as an attribute `'Theta'` to be automatically used in Monte Carlo simulations.

Usage

```
generate_pattern(mo, Theta, df = NULL)
```

Arguments

`mo`	single group object defined by the `mirt` package
`Theta`	a numeric vector indicating the latent theta values for a single person
`df`	(optional) data.frame object containing questions, options, and scoring keys. See `mirtCAT` for details

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

References

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

See Also

`mirtCAT`

Examples

```
## Not run:

# return real response vector given choices and (optional) answers
pat <- generate_pattern(mod, Theta = 0, df=df)
# mirtCAT(df, mo=mod, local_pattern = pat)

# generate single pattern observed in dataset used to define mod
pat2 <- generate_pattern(mod, Theta = 0)
# mirtCAT(mo=mod, local_pattern = pat2)

# generate multiple patterns to be analyzed independently
pat3 <- generate_pattern(mod, Theta = matrix(c(0, 2, -2), 3))
# mirtCAT(mo=mod, local_pattern = pat3)


## End(Not run)
```

`get_mirtCAT_env` *Get the internal working environment state during mirtCAT session*

Description

This function is used to access the internal state of the mirtCAT GUI session. It is only useful when designing a customized GUI using the `shinyGUI$ui` input to `mirtCAT`.

Usage

```
get_mirtCAT_env(sessionName)
```

Arguments

`sessionName`	the name of the session defined in `mirtCAT`

Value

a list containing the internal environmental components for mirtCAT

getPerson *Retrieve person object after running createShinyGUI*

Description

This function returns a suitable person object identical to the result returned by mirtCAT, and is only required when the GUI is launched by the createShinyGUI method.

Usage

```
getPerson()
```

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

References

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

See Also

mirtCAT, mirtCAT_preamble, createShinyGUI

Examples

```
## Not run:

mirtCAT_preamble(df=df)
runApp(createShinyGUI(), port = 8000)

person <- getPerson()
summary(person)

## End(Not run)
```

mirtCAT *Generate an adaptive or non-adaptive test HTML interface*

Description

Provides tools to generate an HTML interface for creating adaptive and non-adaptive educational and psychological tests using the shiny package. Suitable for applying unidimensional and multidimensional computerized adaptive tests using item response theory methodology. Test scoring is performed using the mirt package. However, if no scoring is required (i.e., a standard survey) then defining a mirt object may be omitted.

Usage

```
mirtCAT(
  df = NULL,
  mo = NULL,
  method = "MAP",
  criteria = "seq",
  start_item = 1,
  local_pattern = NULL,
  AnswerFuns = list(),
  design_elements = FALSE,
  cl = NULL,
  progress = FALSE,
  primeCluster = TRUE,
  customTypes = list(),
  design = list(),
  shinyGUI = list(),
  preCAT = list(),
  ...
)

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

## S3 method for class 'mirtCAT'
summary(object, sort = TRUE, ...)

## S3 method for class 'mirtCAT'
plot(
  x,
  pick_theta = NULL,
  true_thetas = TRUE,
  SE = 1,
  main = NULL,
  par.strip.text = list(cex = 0.7),
 par.settings = list(strip.background = list(col = "#9ECAE1"), strip.border = list(col =
```

```
    "black")),
  scales = list(x = list(rot = 90)),
  ...
)
```

Arguments

df — a data.frame containing the character vector inputs required to generate GUI questions through shiny. If factors are supplied instead of character vectors then the inputs will be coerced using the as.character() function (set stringsAsFactors = FALSE when defining a data.frame to avoid this). Each row in the object corresponds to a unique item. The object supports the follow column name combinations as inputs to specify the type of response format, questions, options, answers, and stems:

Type — Indicates the type of response input to use from the shiny package. The supported types are: 'radio' for radio buttons (radioButtons), 'select' for a pull-down box for selecting inputs (selectInput), 'rankselect' for a set of pull-down boxes rank-ordering inputs (selectInput) associated with each option supplied, 'text' and 'textArea' for requiring typed user input (textInput and textAreaInput), 'checkbox' for allowing multiple responses to be checked off (checkboxGroupInput), 'slider' for generating slider inputs (sliderInput), or 'none' for presenting only an item stem with no selection options. Note that slider inputs require additional arguments to be passed; see ... instructions below).

Additionally, if the above types are not sufficient for the desired output then users can create their own response formats and inputs via the customTypes list input (see below). E.g., if a function with the name 'MyTableQuestion' is supplied to customTypes then supplying this type to the df will use this function for the respective item. Note that this is more advanced and requires a working knowledge of shiny's design, inputs, and specifications. This is generally for advanced users to use on an as-per-needed basis.

Question — A character vector containing all the questions or stems to be generated. By default these character vectors are passed to HTML, and therefore allow for HTML tags to be included directly. For example, the following example defines two stems, where the second uses an emphasis tag to provide italics.

```
Question = c('This is the first item stem.', 'This is the <em>second</em> item stem.'))
```

Alternatively, if tag constructor function are preferred these need only be wrapped within a final call to as.character to coerce the shiny.tag expressions into suitable character vectors of HTML code. For example, the above could be expressed as

```
Question = c('This is the first item stem.', as.character(div('This is the', em('second'), 'item stem.')))
```

Moreover, because this input must be a character vector, the use of sapply in concert with as.character can apply this conversion to all elements (often redundantly). Here's an example of this format:

```
Question = sapply(list('This is the first item stem.',
```

```
    div('This is the', em('second'), 'item stem.'),
    div('This is the', strong('third'), br(), br(), 'item stem.'),
    div('Fourth with some code:', code('obj <- 42'))),
        as.character)
```

`Option.#` Names pertaining to the possible response options for each item, where the # corresponds to the specific category. For instance, a test with 4 unique response options for each item would contain the columns (`Option.1`, `Option.2`, `Option.3`, `Option.4`). If, however, some items have fewer categories than others then `NA`'s can be used for response options that do not apply.

`Answer` **or** `Answer.#` (Optional) A character vector (or multiple character vectors) indicating the scoring key for items that have correct answer(s). If there is no correct answer for a question then a value of `NA` must be declared.

Note that 'scoring' some item response data can be ambiguous depending on the stimuli provided, which requires greater attention. For example, when using `'rankselect'`: should partial scoring be used if the ranks are mostly correct; should partial scoring be used if the response are only off by a ranking constant (e.g., correct rank is 1-2-3-4-5, but the respondent ranks 2-3-4-5-1, in which case four relative rankings are correct but 1 is incorrect); should a 0-1 scoring be used to indicate none-all correct?. When this type of ambiguity exists in the multiple-answers cases it is strongly recommended to use the `AnswerFuns` argument instead for better functional control

`Forced` (Optional) logical vector indicating whether the respondent is forced (`TRUE`) or not (`FALSE`) to include a response for the respective item. If omitted from the `df` definition this will be automatically set to `TRUE` for each item. For surveys, it is generally recommended to set this to `FALSE` to allow respondents the ability to not answer questions they may be uncomfortable answering

`Stem` (Optional) a character vector of absolute or relative paths pointing external markdown (.md) or HTML (.html) files to be used as item stems. `NA`s are used if the item has no corresponding file.

`Timer` (Optional) a numeric vector indicating a time limit (in seconds) for each respective item. If a response is not provided before this limit then the question will automatically advance to the next selected item. The values `NA` and `Inf` indicate no time limit for the respective items. Note that this option can only be used when `df$Forced = TRUE`

`Mastery` (Optional) a logical vector indicating whether the item must be mastered prior to continuing. Naturally, this requires that one or more `Answers` are provided, or suitable functions for scoring are supplied

`HTMLOptions` (Optional) a logical vector indicating whether the respective `Option.#` terms should be wrapped within an `HTML` function and rendered for suitable shiny inputs (e.g., radio buttons). This is a short-hand wrapper to the more flexible `choiceNames` approach, which can be used to wrap option inputs with alternative functions.

... In cases where 'slider' inputs are used instead only the Question input is required along with (at minimum) a min, max, and step column. In rows where the Type == 'slider' the column names will correspond to the input arguments to sliderInput. Other input column options such as step, round, pre, post, ticks, inline, placeholder, width, and size are also supported for the respective input types.

mo single group object defined by the mirt::mirt() function. This is required if the test is to be scored adaptively or non-adaptively, but not required for general questionnaires. The object can be constructed by using the generate.mirt_object function if population parameters are known or by including a calibrated model estimated from the mirt function with real data.

method argument passed to mirt::fscores() for computing new scores in the CAT stage, with the addition of a 'fixed' input to keep the latent trait estimates fixed at the previous values. When method = 'ML', if there is no variability in the given response pattern during the CAT (i.e., the participant is responding completely correctly or completely incorrectly) then the method will temporarily be set to MAP until sufficient response variability is present. Default is 'MAP'

criteria adaptive criteria used, default is to administer each item sequentially using criteria = 'seq'.

Possible inputs for unidimensional adaptive tests include: 'MI' for the maximum information, 'MEPV' for minimum expected posterior variance, 'MLWI' for maximum likelihood weighted information, 'MPWI' for maximum posterior weighted information, 'MEI' for maximum expected information, and 'IKLP' as well as 'IKL' for the integration based Kullback-Leibler criteria with and without the prior density weight, respectively, and their root-n items administered weighted counter-parts, 'IKLn' and 'IKLPn'.

Possible inputs for multidimensional adaptive tests include: 'Drule' for the maximum determinant of the information matrix, 'Trule' for the maximum (potentially weighted) trace of the information matrix, 'Arule' for the minimum (potentially weighted) trace of the asymptotic covariance matrix, 'Erule' for the minimum value of the information matrix, and 'Wrule' for the weighted information criteria. For each of these rules, the posterior weight for the latent trait scores can also be included with the 'DPrule', 'TPrule', 'APrule', 'EPrule', and 'WPrule', respectively.

Applicable to both unidimensional and multidimensional tests are the 'KL' and 'KLn' for point-wise Kullback-Leibler divergence and point-wise Kullback-Leibler with a decreasing delta value (delta*sqrt(n), where n is the number of items previous answered), respectively. The delta criteria is defined in the design object

Non-adaptive methods applicable even when no mo object is passed are: 'random' to randomly select items, and 'seq' for selecting items sequentially.

start_item two possible inputs to determine the starting item are available. Passing a number will indicate the specific item to be used as the start item; default is 1, which selects the first item in the defined test/survey. If a character string is passed then the item will be selected from one of the item selections criteria available (see the criteria argument). For off-line runs where a local_pattern input is

used then a vector of numbers/characters may be supplied and will be associated with each row response vector

`local_pattern` a character/numeric matrix of response patterns used to run the CAT application without generating the GUI interface. This option requires complete response pattern(s) to be supplied. `local_pattern` is required to be numeric if no `questions` are supplied, and the responses must be within a valid range of the defined `mo` object. Otherwise, it must contain character values of plausible responses which corresponds to the answer key and/or options supplied in `df`. If the object contains an attribute `'Theta'` then these values will be stored within the respective returned objects. See `generate_pattern` to generate response patterns for Monte Carlo simulations

`AnswerFuns` a list with the length equal to the number of items in the item bank consisting of user-defined functions. These functions are used to determine whether a given response obtained from the GUI is 'correct' or 'incorrect' by returning a logical scalar value, while `NA`'s must be used to indicate `AnswerFuns` should not be used for a given item. Note that `AnswerFuns` is given priority over the answers provided by `df`, therefore any answers provided by `df` will be entirely ignored.

For example, the following provides a customized response function for the first item.

```
AnswerFuns <- as.list(rep(NA, nrow(df)))
AnswerFuns[[1]] <- function(input) input == '10' || to.lower(input) == 'ten'
```

`design_elements`

logical; return an object containing the test, person, and design elements? Primarily this is to be used with the `findNextItem` function

`cl` an object definition to be passed to the parallel package (see `?parallel::parLapply` for details). If defined, and if `nrow(local_pattern) > 1`, then each row will be run in parallel to help decrease estimation times in simulation work

`progress` logical; print a progress bar to the console with the `pbapply` package for given response patterns? Useful for gauging how long Monte Carlo simulations will take to finish

`primeCluster` logical; when a `cl` object is supplied, should the cluster be primed first before running the simulations in parallel? Setting to `TRUE` will ensure that using the cluster will be optimal every time a new `cl` is defined. Default is `TRUE`

`customTypes` an optional list input containing functions for Designing Original Graphical Stimuli (DOGS). DOGS elements in the input list must contain a unique name, and the item with which it is associated must be declared in the a `df$Type` input. The functions defined must be of the form

```
myDOGS <- function(inputId, df_row) ...
```

and must return, at the very minimum, an associated `shiny` input object that makes use of the `inputId` argument (e.g., `radioButtons`). Any valid shiny object can be returned, including lists of shiny objects. As well, the `df_row` argument contains any extra information the users wishes to obtain from the associated row in the `df` object.

The following is a simple example of DOGS for a true-false question and how it is passed:

```
good_dogs <- function(inputId, df_row){
   return(list(h2('This statement is false'),
               radioButtons(inputId = inputId, label='',
                     choices = c('True', 'False'), selected = '')
        ))
   }

df <- data.frame(Question = '', ..., Type = 'Doug')
results <- mirtCAT(df=df, customTypes = list(Doug = good_dogs))
```

IMPORTANT: When using the custom inputs the select defined `Type` must be unique, even when the function defined (e.g., `good_dog` above) is recycled. Hence, if two items were to use the `good_dog` function then `df` should be defined as something like `df$Type <- c('Doug1', 'Doug2')` with the associated `customTypes = list(Doug1=good_dog, Doug2=good_dog)`

`design` a list of design based control parameters for adaptive and non-adaptive tests. These can be

`min_SEM` Default is `rep(0.3, nfact)`; minimum standard error or measurement to be reached for the latent traits (thetas) before the test is stopped. If the test is multidimensional, either a single value or a vector of values may be supplied to provide SEM criteria values for each dimension

`delta_thetas` Default is `rep(0, nfact)`; stopping criteria based on the change in latent trait values (e.g., a change from `theta = 1.5` to `theta = 1.54` would stop the CAT if `delta_thetas = 0.05`). The default disables this stopping criteria

`thetas.start` a numeric vector of starting values for the theta parameters (default is `rep(0, nfact)`) or an `matrix` with N rows and nfact columns, where N is equal to `nrow(local_pattern)`

`min_items` minimum number of items that must be answered before the test is stopped. Default is `1`

`max_items` maximum number of items that can be answered. Default is the length of the item bank

`max_time` maximum time allowed for the generated GUI, measured in seconds. For instance, if the test should stop after 10 minutes then the number 600 should be passed (10 * 60). Default is `Inf`, therefore no time limit

`quadpts` Number of quadrature points used per dimension for integration (if required). Default is identical to scheme in `fscores`

`theta_range` upper and lower range for the theta integration grid. Used in conjunction with `quadpts` to generate an equally spaced quadrature grid. Default is `c(-6,6)`

`allow_constrain_breaks` logical; should the test be allowed to terminate in the middle of administering the items in an (un)ordered testlet set specified in `constraints`? Default is `FALSE`

`weights` weights used when `criteria == 'Wrule'`, but also will be applied for weighted trace functions in the T- and A-rules. The default weights the latent dimensions equally. Default is `rep(1, nfact)`, where `nfact` is the number of test dimensions

KL_delta interval range used when criteria = 'KL' or criteria = 'KLn'. Default is 0.1

content an optional character vector indicating the type of content measured by an item. Must be supplied in conjunction with content_prop

content_prop an optional named numeric vector indicating the distribution of item content proportions. A content vector must also be supplied to indicate the item content membership. For instance, if content contains three possible item content domains 'Addition', 'Subtraction', and 'Multiplication', and the test should contain approximately half multiplication and a quarter of both addition and subtraction, then a suitable input would be content_prop = c('Addition'=0.25, 'Subtraction'=0.25, 'Multiplication'=.5) Note that content_prop must sum to 1 in order to represent valid population proportions.

classify a numeric vector indicating cut-off value(s) for classification above or below some prior threshold. Default does not use the classification scheme

classify_CI a numeric vector indicating the confident intervals used to classify individuals being above or below values in classify. Values must be between 0 and 1 (e.g., 0.95 gives 95% confidence interval)

sprt_lower a numeric vector indicating lower cut-off value(s) for classification above or below some prior threshold using the sequential probability ratio test. Default does not use the classification scheme

sprt_upper a numeric vector indicating upper cut-off value(s) for classification above or below some prior threshold using the sequential probability ratio test. Default does not use the classification scheme

sprt_alpha a numeric vector indicating the lower-bound error rate to use for SPRT. Default is .05

sprt_beta a numeric vector indicating the upper-bound error rate to use for SPRT. Default is .05

exposure a numeric vector specifying the amount of exposure control to apply for each successive item (length must equal the number of items). Note that this includes the first item as well when a selection criteria is specified, therefore if a specific first item should be used then the first element to exposure should be 1. The default uses no exposure control.

If the item exposure is greater than 1 then the n most optimal criteria will be randomly sampled from. For instance, if exposure[5] == 3, and criteria = 'MI', then when the fifth item is to be selected from the remaining pool of items the top 3 candidate items demonstrating the largest information criteria will be sampled from. Naturally, the first and last elements of exposure are ignored since exposure control will be meaningless.

If all elements in exposure are between 0 and 1 then the Sympson-Hetter exposure control method will be implemented. In this method, an item is administered only if it passes a probability simulation experiment; otherwise, it is removed from the item pool. Values closer to 1 are more likely to appear in the test, while value closer to 0 are more likely to be randomly discarded.

constraints A named list declaring various item selection constraints for which particular item, where each list element is a vector of item numbers. Unless

otherwise stated, multiple elements can be declared (e.g., `list(ordered = c(1:5), ordered = c(7:9))` is perfectly acceptable). These include:

`not_scored` declaring items that can be selected but will not be used in the scoring of the CAT. This is primarily useful when including experimental items for future CATs. Only one vector of `not_scored` elements can be supplied

`excluded` items which should not actually appear in the session (useful when re-testing participants who have already seen some of the items). Only one vector of `excluded` elements can be supplied

`independent` declaring which items should never appear in the same CAT session. Use this if, for example, item 1 and item 10 have very similar questions types and therefore should not appear within the same session

`ordered` if one item is selected during the CAT, administer this particular group of items in order according to the specified sequence

`unordered` same as ordered, except the items in the group will be selected at random until the group is complete

`customUpdateThetas` a more advanced function of the form `customUpdateThetas <- function(design, person, test)` to update the ability/latent trait estimates throughout the CAT (or more generally, scoring) session. The `design`, `person`, and `test` are the same as in `customNextItem`. The latent trait terms are updated directly in the `person` object, which is a `ReferenceClasses` type, and therefore direct assignment to the object will modify the internal elements. Hence, to avoid manual modification users can pass the latent trait estimates and their respective standard errors to the associated `person$Update_thetas(theta, theta_SE)` function. Note that the `fscores()` function can be useful here to capitalize on the estimation algorithms implemented in `mirt`.

For example, a minimal working function would look like the following (note the use of `rbind()` to append the history terms in the `person` object):

```
myfun <- function(design, person, test){
    mo <- extract.mirtCAT(test, 'mo')
    responses <- extract.mirtCAT(person, 'responses')
    tmp <- fscores(mo, response.pattern = responses)
    person$Update_thetas(tmp[,'F1'],
                         tmp[,'SE_F1', drop=FALSE])
    invisible()
 }
```

`customNextItem` a more advanced function of the form `customNextItem <- function(design, person, test)` to use a customized item selection method. This requires more complex programming and understanding of `mirtCAT`s internal elements, and it's recommended to initially use a `browser` to understand the state of the input arguments. When defined, all but the `not_scored` input to the optional `constraints` list will be ignored.

Use this if you wish to program your item selection techniques explicitly, though this can be combined the internal `findNextItem` function with analogous inputs. Function must return a single integer value indicating the next

item to administer or an NA value to indicate that the test should be terminated. See extract.mirtCAT for details on how to extract and manipulate various internal elements from the required functional arguments

constr_fun (WARNING: supplying this function will disable a number of the heuristic item selection constraints in the constraints list as a consequence; namely, all list options except for "not_scored").
This argument contains an optional user-defined function of the form function(design, person, test) that returns a data.frame containing the left hand side, relationship, and right hand side of the constraints for lp. Each row corresponds to a constraint, while the number of columns should be equal to the number of items plus 2. Note that the column names of the returned data.frame object do not matter.
For example, say that for a given test the user wants to add the constraint that exactly 10 items should be administered to all participants, and that items 1 and 2 should not be included in the same test. The input would then be defined as

```
const_fun <- function(design, person, test){
       nitems <- extract.mirt(test@mo, 'nitems')
       lhs <- matrix(0, 2, nitems)
       lhs[1, ] <- 1
       lhs[2, c(1,2)] <- 1
     data.frame(item=lhs, relation=c("==", "<="), value=c(10, 1))
     }
```

The definition above corresponds to the constraints 1 * x1 + 1 * x2 + ... + 1 * xn = 10 and 1 * x1 + 1 * x2 + 0 * x3 + ... + 0 * xn <= 1 , where the x terms represent binary indicators for each respective item which the optimizer is searching through. Given some objective vector supplied to findNextItem, the most optimal 10 items will be selected which satisfy these two constraints, meaning that 1) exactly 10 items will be administered, and 2) if either item 1 or 2 were selected these two items would never appear in the same test form (though neither is forced to appear in any given test). See findNextItem for further details and examples

test_properties a user-defined data.frame object to be used with a supplied customNextItem function. This should be used to define particular properties inherent to the test items (e.g., whether they are experimental, have a particular weighting scheme, should only be used for one particular group of individuals, and so on). The number of rows must be equal to the number of items in the item bank, and each row corresponds to the respective item. This input appears within the internal design object in a test_properties slot.

person_properties a user-defined data.frame object to be used with a supplied customNextItem function. This should be used to define particular properties inherent to the individuals participants (e.g., known grouping variable, age, whether they've taken the test before (and which items they took), and so on). In off-line simulations, the number of rows must be equal to the number of participants. This input appears within the internal design object in a person_properties slot; for Monte Carlo simulations, rows should be manually indexed using the person$ID slot.

shinyGUI a list of GUI based parameters to be over-written. These can be

`title` A character string for the test title. Default is `'mirtCAT'`

`authors` A character string for the author names. Default is `'Author of survey'`. If the input is an empty string (`''`) then the author information will be omitted in the GUI

`instructions` A two part character vector indicating how to use the GUI. Default is:

```
c("To progress through the interface, click on the action button below.",
       "Next")
```

The second part of the character vector provides the name for the action button.

`itemtimer` A character string to display the item-timer clock. Default is `'Item timer: '`

`incorrect` A character string to display in case of a failed response. Default is `'The answer provided was incorrect. Please select an alternative.'`

`failpass` A character string to display in case of a failed password input. Default is `'Incorrect Login Name/Password. Please try again (you have %s attempts remaining).'`

`timemsg` A three part character vector indicating words for hour, minute, second & and. Default is `c('hour ','minutes ','seconds ', 'and ')`

`firstpage` The first page of the shiny GUI. Default prints the title and information message.

```
list(h1('Welcome to the mirtCAT interface'),
 sprintf('The following interface was created using the mirtCAT package v
 To cite the package use citation(\'mirtCAT\') in R.',
        packageVersion("mirtCAT")))
```

If an empty list is passed, this page will be skipped.

`begin_message` Text to display on the page prior to beginning the CAT. Default is `"Click the action button to begin."` for scored tests whereby a `mo` object has been include, while the default is `""` for non-scored tests (which disables the page).

`demographics` A person information page used in the GUI for collecting demographic information, generated using tools from the shiny package. For example, the following code asks the participants about their Gender:

```
list(selectInput(inputId = 'gender',
         label = 'Please select your gender.',
        choices = c('', 'Male', 'Female', 'Other'),
         selected = ''))
```

By default, the demographics page is not included.

`demographics_inputIDs` a character vector required if a custom demographics input is used. Default is `demographics_inputIDs = 'gender'`, corresponding to the `demographics` default

`stem_default_format` `shiny` function used for the stems of the items. Default uses the `HTML` wrapper, allowing for HTML tags to be included directly in

the character vector definitions. To change this to something different, like h5 for example, pass stem_default_format = shiny::h5 to the shinyGUI list

temp_file a character vector indicating where a temporary .rds file containing the response information should be saved while the GUI is running. The object will be saved after each item is successfully completed. This is used to save response information to the hard drive in case there are power outages or unexpected computer restarts.
If NULL, no temp file will be created. Upon completion of the test, the temp file will be deleted. If a file already exists, however, then this will be used to resume the GUI at the last location where the session was interrupted

lastpage A function printing the last message, indicating that the test has been completed (i.e., criteria has been met). The function requires exactly one argument (called person), where the input argument is the person object that has been updated throughout the test. The default function is

```
function(person){
              return(list(h5("You have successfully completed the interface.
                        It is now safe to leave the application.")))
                    }
```

css a character string defining CSS elements to modify the GUI presentation elements. The input string is passed to the argument tags$style(HTML(shinyGUI$css)) prior to constructing the user interface

theme a character definition for the shinytheme package to globally change the GUI theme

choiceNames a list containing the choiceNames input for each respective item when the input is 'radio' or 'checkbox' (see radioButtons), where each element is itself a list of instructions.
This is used to modify the output of the controllers using suitable HTML code. If a row in df should not have a customized names then supplying the value NULL in the associated list element will use the standard inputs instead. Alternatively, if specified the names of the elements to this list can be used to match the rownames of the df object to avoid the use of NULL placeholders

choiceValues associated values to be used along with choiceNames (see above)

time_before_answer a numeric value representing the number of seconds that must have elapsed when df$Forced = FALSE before a response can be provided or skipped. This is used to control accidental skips over items when responses are not forced. Default is 1, indicating one full second

password a data.frame object indicating the user name (optional) and password required prior to beginning the CAT. Possible options are

No User Information a single row data.frame. Each column supplied in this case will be associated with a suitable password for all individuals. Naturally, if only 1 column is defined then there is only 1 global password for all users

User Information Pairing a multi-row data.frame where the first column represents the user name and all other columns as the same as the first option. E.g., if two users ('name1' and 'name2') are given the same

password '1234' then `password = data.frame(User = c('user1', 'user2'), Password = rep('1234', 2))`

`response_msg` string to print when valid responses are required but the users does not provide a valid input. Default is `"Please provide a suitable response"`

`ui` a shiny UI function used to define the interface. If `NULL`, the default one will be used. See `mirtCAT:::default_UI` for the internal code definition

`preCAT` a list object which can be used to specify a pre-CAT block in which different test properties may be applied prior to beginning the CAT session. If the list is empty, no preCAT block will be used. All of the following elements are required to use the `preCAT` input:

`min_items` minimum number of items to administer before the CAT session begins. Default is 0

`max_items` max number of items to administer before the CAT session begins. An input greater than 0 is required to run the preCAT stage

`criteria` selection criteria (see above). Default is 'random'

`method` estimation criteria (see above). It is generally recommended to select a method which can deal with all-or-none response patterns, such as 'EAP', 'MAP', or 'WLE'. Default is 'MAP'

`response_variance` logical; terminate the preCAT stage when there is variability in the response pattern (i.e., when maximum-likelihood estimation contains a potential optimum)? Default is FALSE

`...` additional arguments to be passed to `mirt`, `fscores`, `runApp`, or `lattice`

`x` object of class `'mirtCAT'`

`object` object of class `'mirtCAT'`

`sort` logical; sort the response patterns based on the order they were administered? If FALSE, the raw response patterns containing NAs will be returned for items that were not administered

`pick_theta` a number indicating which theta to plot (only applicable for multidimensional tests). The default is to facet each theta on one plot, but to plot only the first factor pass `pick_theta = 1`

`true_thetas` logical; include a horizontal line indicating where the population-level theta values are? Only applicable to Monte Carlo simulations because this value would not be known otherwise

`SE` size of the standard errors to plot. The default is 1, and therefore plots the standard error. To obtain the 95% interval use `SE = 1.96` (from the z-distribution)

`main` title of the plot. Will default to `'CAT Standard Errors'` or `'CAT ##% Confidence Intervals'` depending on the SE input

`par.strip.text` plotting argument passed to `lattice`

`par.settings` plotting argument passed to `lattice`

`scales` plotting argument passed to `lattice`

Details

All tests will stop once the 'min_SEM' criteria has been reached or classification above or below the specified cutoffs can be made. If all questions should be answered, users should specify an extremely small 'min_SEM' or, equivalently, a large 'min_items' criteria to the design list input.

Value

Returns a list object of class 'Person' containing the following elements:

raw_responses A character vector indicating the raws responses to the respective items, where NA indicates the item was not answered

scored_responses An integer vector of scored responses if the item_answers input was used for each respective item

items_answered An integer vector indicating the order in which the items were answered

thetas A numeric vector indicating the final theta estimates

SE_thetas A numeric vector indicating the standard errors of the final theta estimates

thetas_history A matrix indicating the progression of updating the theta values during the test

thetas_SE_history A matrix indicating the standard errors for theta after each successive item was answered

item_time A numeric vector indicating how long the respondent took to answer each question (in seconds)

demographics A data.frame object containing the information collected on the first page of the shiny GUI. This is used to store the demographic information for each participant

classification A character vector indicating whether the traits could be classified as 'above' or 'below' the desired cutoffs

HTML help files, exercises, and examples

To access examples, vignettes, and exercise files that have been generated with knitr please visit https://github.com/philchalmers/mirtCAT/wiki.

Modifying the design object directly through customNextItem() (advanced)

In addition to providing a completely defined item-selection map via the customNextItem() function, users may also wish to control some of the more fine-grained elements of the design object to adjust the general control parameters of the CAT (e.g., modifying the maximum number of items to administer, stopping the CAT if something peculiar has been detected in the response patterns, etc). Note that this feature is rarely required for most applications, though more advanced users may wish to modify these various low-level elements of the design object directly to change the flow of the CAT to suit their specific needs.

While the person object is defined as a Reference Class (see setRefClass) the design object is generally considered a fixed S4 class, meaning that, unlike the person object, it's elements are not mutable. Therefore, in order to make changes directly to the design object the users should follow these steps:

1. Within the defined `customNextItem` function, the `design` object slots are first modified (e.g., `design@max_items <- 20L`).
2. Along with the desired next item scalar value from `customNextItem()`, the scalar object should also contain an attribute with the name `'design'` which holds the newly defined `design` object (e.g., `attr(ret, 'design') <- design; return(ret)`).

Following the above process the work-flow in `mirtCAT` will use the new `design` object in place of the old one, even in Monte Carlo simulations.

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

References

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

See Also

`generate_pattern`, `generate.mirt_object`, `extract.mirtCAT`, `findNextItem`, `computeCriteria`

Examples

```
## Not run:

### unidimensional scored example with generated items

# create mo from estimated parameters
```

```
set.seed(1234)
nitems <- 50
itemnames <- paste0('Item.', 1:nitems)
a <- matrix(rlnorm(nitems, .2, .3))
d <- matrix(rnorm(nitems))
dat <- simdata(a, d, 1000, itemtype = 'dich')
mod <- mirt(dat, 1)
coef(mod, simplify=TRUE)

# alternatively, define mo from population values (not run)
pars <- data.frame(a1=a, d=d)
mod2 <- generate.mirt_object(pars, itemtype='2PL')
coef(mod2, simplify=TRUE)

# simple math items
questions <- answers <- character(nitems)
choices <- matrix(NA, nitems, 5)
spacing <- floor(d - min(d)) + 1 #easier items have more variation in the options

for(i in 1:nitems){
    n1 <- sample(1:50, 1)
    n2 <- sample(51:100, 1)
    ans <- n1 + n2
    questions[i] <- paste0(n1, ' + ', n2, ' = ?')
    answers[i] <- as.character(ans)
    ch <- ans + sample(c(-5:-1, 1:5) * spacing[i,], 5)
    ch[sample(1:5, 1)] <- ans
    choices[i, ] <- as.character(ch)
}

df <- data.frame(Question=questions, Option=choices,
                              Type = 'radio', stringsAsFactors = FALSE)
head(df)

(res <- mirtCAT(df)) #collect response only (no scoring or estimating thetas)
summary(res)

# include scoring by providing Answer key
df$Answer <- answers
(res_seq <- mirtCAT(df, mod)) #sequential scoring
(res_random <- mirtCAT(df, mod, criteria = 'random')) #random
(res_MI <- mirtCAT(df, mod, criteria = 'MI', start_item = 'MI')) #adaptive, MI starting item

summary(res_seq)
summary(res_random)
summary(res_MI)

#----------------------------------------
# HTML tags for better customization, coerced to characters for compatibility

# help(tags, package='shiny')
options <- matrix(c("Strongly Disagree", "Disagree", "Neutral", "Agree", "Strongly Agree"),
                  nrow = 3, ncol = 5, byrow = TRUE)
```

```
shinyStems <- list(HTML('Building CATs with mirtCAT is difficult.'),
            div(HTML('mirtCAT requires a'), br(), HTML('substantial amount of coding.')),
               div(strong('I would use'), HTML('mirtCAT in my research.')))
questions <- sapply(shinyStems, as.character)
df <- data.frame(Question=questions,
                 Option = options,
                 Type = "radio",
                 stringsAsFactors=FALSE)

res <- mirtCAT(df)
res

#---------------------------------------

# run locally, random response pattern given Theta
set.seed(1)
pat <- generate_pattern(mod, Theta = 0, df=df)
head(pat)

# seq scoring with character pattern for the entire test (adjust min_items)
res <- mirtCAT(df, mod, local_pattern=pat, design = list(min_items = 50))
summary(res)

# same as above, but using special input vector that doesn't require df input
set.seed(1)
pat2 <- generate_pattern(mod, Theta = 0)
head(pat2)
print(mirtCAT(mo=mod, local_pattern=pat2))

# run CAT, and save results to object called person (start at 10th item)
person <- mirtCAT(df, mod, item_answers = answers, criteria = 'MI',
                  start_item = 10, local_pattern = pat)
print(person)
summary(person)

# plot the session
plot(person) #standard errors
plot(person, SE=1.96) #95 percent confidence intervals

#---------------------------------------

### save response object to temp directory in case session ends early
wdf <- paste0(getwd(), '/temp_file.rds')
res <- mirtCAT(df, mod, shinyGUI = list(temp_file = wdf))

# resume test this way if test was stopped early (and temp files were saved)
res <- mirtCAT(df, mod, shinyGUI = list(temp_file = wdf))
print(res)


## End(Not run)
```

mirtCAT_preamble *Preamble function called by mirtCAT*

Description

This is largely an internal function called by mirtCAT, however it is made public for better use with external web-hosting interfaces (like shinyapps.io). For more information see https://shiny.rstudio.com/articles/persistent-data-storage.html for further information about saving output remotely when using shiny.

Usage

```
mirtCAT_preamble(..., final_fun = NULL)
```

Arguments

...	arguments passed to mirtCAT
final_fun	a function called just before the shiny GUI has been terminated, primarily for saving results externally with packages such as rDrop2, RAmazonS3, googlesheets, RMySQL, personal servers, and so on when applications are hosted on the web. The function must be of the form final_fun <- function(person){...}, where person is the standard output returned from mirtCAT

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

References

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

See Also

mirtCAT, createShinyGUI, getPerson

Examples

```
## Not run:

mirtCAT_preamble(df = df)


## End(Not run)
```

updateDesign *Update design elements*

Description

A function that will update the object returned from findNextItem. This can be used to run the CAT session manually in a set-by-step manner.

Usage

```
updateDesign(x, new_item, new_response, updateTheta = TRUE)
```

Arguments

x	an object of class 'mirtCAT_design' returned from the mirtCAT function when passing design_elements = TRUE
new_item	a numeric vector indicating which items to select
new_response	a numeric vector indicating the responses the the selected items
updateTheta	logical; update the internal ability terms after the new item response has been added to the internal objects?

Value

returns an object of class 'mirtCAT_design' with updated elements.

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

References

Chalmers, R., P. (2012). mirt: A Multidimensional Item Response Theory Package for the R Environment. *Journal of Statistical Software, 48*(6), 1-29. doi:10.18637/jss.v048.i06

Chalmers, R. P. (2016). Generating Adaptive and Non-Adaptive Test Interfaces for Multidimensional Item Response Theory Applications. *Journal of Statistical Software, 71*(5), 1-39. doi:10.18637/jss.v071.i05

See Also

mirtCAT, findNextItem

Examples

```
## Not run:

set.seed(1)
nitems <- 100
itemnames <- paste0('Item.', 1:nitems)
a <- matrix(rlnorm(nitems, .2, .3))
d <- matrix(rnorm(nitems))
dat <- simdata(a, d, 500, itemtype = 'dich')
colnames(dat) <- itemnames
mod <- mirt(dat, 1, verbose = FALSE)

# test defined in mirtCAT help file, first example
CATdesign <- mirtCAT(mo = mod, criteria = 'MI', design_elements = TRUE,
  start_item = 2)

# returns 2 in this case, since that was the starting item
findNextItem(CATdesign)

# first iteration, no answered items
CATdesign$person$items_answered

# update when next item is item 2 and answered correctly
CATdesign <- updateDesign(CATdesign, new_item = 2, new_response = 1)
CATdesign$person$items_answered  # item 2 answered first
CATdesign$person$responses       # in item 2 element response was = 1
CATdesign$person$thetas # current estimate
findNextItem(CATdesign)

# determine next item if item 70 were also answered correctly next
CATdesign <- updateDesign(CATdesign, new_item = 70, new_response = 1)
CATdesign$person$items_answered
CATdesign$person$responses
findNextItem(CATdesign)

# continue on, now with item 95 added next (answered incorrectly)
CATdesign <- updateDesign(CATdesign, new_item = 95, new_response = 0)
CATdesign$person$thetas
CATdesign$person$thetas_history
CATdesign$person$thetas_SE_history
findNextItem(CATdesign)


## End(Not run)
```

Index

∗ **adaptive**
    mirtCAT, 19
∗ **computerized**
    mirtCAT, 19
∗ **package**
    mirtCAT-package, 2
∗ **testing**
    mirtCAT, 19

as.character, *20*

browser, *26*

checkboxGroupInput, *20*
computeCriteria, 3, *10*, *32*
createSessionName, 5
createShinyGUI, 5, *18*, *35*

extract.mirt, *8*
extract.mirtCAT, *4*, 6, *11*, *27*, *32*

findNextItem, *4*, *8*, 10, *23*, *26, 27*, *32*, *36*
fscores, *24*, *30*

generate.mirt_object, 14, *22*, *32*
generate_pattern, *15*, 16, *23*, *32*
get_mirtCAT_env, 17
getPerson, *6*, 18, *35*

h5, *29*
HTML, *20, 21*, *28*

lattice, *30*
lp, *11*, *27*

mirt, *8*, *14, 15*, *19*, *22*, *30*
mirtCAT, *3–6*, *8*, *10, 11*, *14–18*, 19, *32*, *35, 36*
mirtCAT-package, 2
mirtCAT_preamble, *5, 6*, *18*, 35

plot.mirtCAT *(*mirtCAT*)*, 19
print.mirtCAT *(*mirtCAT*)*, 19

radioButtons, *20*, *23*, *29*
ReferenceClasses, *26*
runApp, *5*, *30*

sapply, *20*
selectInput, *20*
setRefClass, *31*
shiny, *19*
shinyApp, *5*
sliderInput, *20*, *22*
stopApp, *5*
summary.mirtCAT *(*mirtCAT*)*, 19

textAreaInput, *20*
textInput, *20*

updateDesign, *4*, *10, 11*, 36