Package ‘conquestr’

May 18, 2025

Type Package

Title An R Package to Extend 'ACER ConQuest'

Version 1.5.1

Description Extends 'ACER ConQuest' through a family of functions designed to improve graphical outputs and help with advanced analysis (e.g., differential item functioning). Allows R users to call 'ACER ConQuest' from within R and read 'ACER ConQuest' System Files (generated by the command \`put\` <https://conquestmanual.acer.org/s4-00.html#put>). Requires 'ACER ConQuest' version 5.40 or later. A demonstration version can be downloaded from <https://shop.acer.org/acer-conquest-5.html>.

URL https://www.acer.org/au/conquest, https://conquestmanual.acer.org, https://shop.acer.org/acer-conquest-5.html

Imports dplyr, ggplot2 (>= 3.5.1), ggrepel, kableExtra, magrittr, methods, Rcpp, rlang, stats, stringr, tidyr, tidyselect, zlib

Suggests knitr, gridExtra, rmarkdown, testthat (>= 3.2.1)

LinkingTo Rcpp

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

SystemRequirements ACER ConQuest (>=5.40.0)

Collate \`\`RcppExports.R'' \`\`ReadConQuestLibrary.R'' \`\`ReadConQuestRout_createDF.R'' \`\`ReadConQuestState.R'' \`\`postProcessCqSysfile.R'' \`\`ReadConQuestState_createDF.R'' \`\`conquestr.R'' \`\`conquestrFunc.R'' \`\`generateItems.R'' \`\`generateHelpers.R'' \`\`infoHelpers.R'' \`\`itanalHelpers.R'' \`\`plotGeneral.R'' \`\`plotRout.R'' \`\`pvHelpers.R'' \`\`residHelpers.R'' \`\`showHelpers.R'' \`\`thrstThrsh.R'' \`\`cleaningHelpers.R''

Depends R (>= 4.1.0)

NeedsCompilation yes

Author Dan Cloney [aut, cre] (ORCID: <`https://orcid.org/0000-0002-2130-237X`>),
Ray Adams [aut]

Maintainer Dan Cloney <`dan.cloney@acer.org`>

Repository CRAN

Date/Publication 2025-05-18 07:50:02 UTC

Contents

`checkItemRespValid` *checkItemRespValid*

Description

Check that the item responses in raw data are: (1) valid, (2) each valid response mapped to an item appears at least once, and (3) each valid reponse mapped to an item has sufficently many responses (defaults to a minimum of 10 observations for each response category)

Usage

```
checkItemRespValid(data, caseID, validMap, varLabel, validLabel)
```

Arguments

`data`	Raw data, a data frame.
`caseID`	A string indicating the name of the case id variable in the data.
`validMap`	A data frame which contains a mapping of valid responses to item lables. This data frame shoudl be in long format, with each valis response * item conbination representing a row.
`varLabel`	A string indicating the name of the variable in validMap that identidies the valis items names/lables.
`validLabel`	A string indicating the name of the variable in validMap that contains the valid codes/responses for each item. This shoudl include missing values (e.g., "99")

Value

A list of lists: one list per item in validMap$varLabel. Within each list, there can be up to three dfs: (1) the case ids and invalid responses for the item, (2) the valid codes not observed in the data set, and (3) the valid codes observed fewer than 10 times in the data. NOTE: a wanring is thrown if the validMap$varLabel is not found in the data.

`checkVars` *checkVars*

Description

Check raw data: are all required variables preseent and ensure there are no extraneous variables.

Usage

```
checkVars(data, varNames, except = NULL)
```

Arguments

`data`	Raw data, a data frame.
`varNames`	Vector of valid variable names.
`except`	A vector of variable names to be excluded form the check.

Value

A list.

`cnvrtItemParam` *cnvrtItemParam*

Description

takes an item in one model's parameterisation and returns it in another parameterisation.

Usage

```
cnvrtItemParam(item, from = "muraki", to = "conquest", D = 1)
```

Arguments

`item`	an item design matrix that is of size response categories (m) by four: • column one is category values, usually from 0 to m. Sometimes referred to as 'x', and in this case, this value times the discrimination is the category score. • column two is the delta dot parameter repeated m times (the average difficulty of the item) • column three is the tau (step) parameter where for the first response category (x = 0) tau = 0, and for m >= 2, entries are deviations from delta dot. In the dichotomous case, all items in this column are zero. • column four is the discrimination parameter ("a")
`from`	a string, either "muraki" or "conquest" (default) (see 10.1177/0146621697211001). Describing the parameterisation of *item*
`to`	a string, either "muraki" or "conquest" (default) (see 10.1177/0146621697211001). Describing the output parameterisation of the returned *item* parameter matrix Note that "muraki" assumes the scaling constant D = 1.7 (to give the normal ogive metric)
`D`	a number, giving the scaling constant. Default is 1 (logistic metric). Other common values are D = 1.7 (to give the normal ogive metric)

Value

an m x 4 matrix of item parameters. The same dimensions as the input, *item*

Examples

```
myTheta <- 0
myDelta <- 1.5
a <- 1.5
m <- 3
itemParamX <- seq(0, m-1, 1)
itemParamD <- rep(myDelta, m)
itemParamT <- c(0, -0.5, 0.5)
itemParamA <- rep(a, m)
itemParam <- cbind(itemParamX, itemParamD, itemParamT, itemParamA)
colnames(itemParam)<- c("x", "d", "t", "a")
myItem <- cnvrtItemParam(itemParam, from = "conquest", to = "muraki")
```

`ConQuestCall` *ConQuestCall*

Description

Call an instance of 'ACER ConQuest' at the command line and run a control file (syntax).

Usage

```
ConQuestCall(cqc, cqExe, stdout = "")
```

Arguments

`cqc`	The location of the control file (syntax) to be run.
`cqExe`	The path to the 'ACER ConQuest' executable. Note, if this argument is missing, conquestr will find a local installation of ACER ConQuest by first searching the default installation locations (Program Files on Windows and Applications on Mac) then searching other local directories (Appdata and the HOME path).
`stdout`	On Mac only, can be toggled to NULL (or a connection) to suppress output to R console.

Value

prints 'ACER ConQuest' output to stdout.

Examples

```
## Not run:
ConQuestCall()

## End(Not run)
```

`ConQuestRout` *ConQuestRout*

Description

Read an 'ACER ConQuest' rout file (a binary file) created by a `plot` command in 'ACER ConQuest'.

Usage

```
ConQuestRout(myRout)
```

Arguments

`myRout`	The path to the binary rout file.

Value

A list containing the data objects to recreate 'ACER ConQuest' plots.

Examples

```
myPlot <- ConQuestRout()
## Not run:
# the default example is an ICC plot from Example 1
# <https://conquestmanual.acer.org/s2-00.html#s2-02>.
str(myPlot)

## End(Not run)
```

`ConQuestSys` *ConQuestSys*

Description

Read an ”ACER ConQuest” system file created by a `put` command in ’ACER ConQuest’. The system file must not be compressed. Use the option ‘compressed=no“ in the put command within ’ACER ConQuest’.

Usage

```
ConQuestSys(myCqs, isMini = FALSE)
```

Arguments

`myCqs`	The location of an uncompressed ’ACER ConQuest’ system file created by ’ACER ConQuest’ > 4.35.
`isMini`	A boolean, set to TRUE if the system file is a *mini* system file created by ’ACER ConQuest’ command put with option "mini = yes".

Value

A list containing the data objects created by ’ACER ConQuest’.

Examples

```
mySysData <- ConQuestSys()
myEx1SysData <- ConQuestSys(myCqs = system.file("extdata", "mysysfile.cqs", package = "conquestr"))
## Not run:
# if you run the above example this will return your original 'ACER ConQuest' syntax.
cat(unlist(myEx1SysData$gCommandHistory))

## End(Not run)
```

`createConQuestProject` *createConQuestProject*

Description

creates a standard folder structure to work with ’ACER ConQuest’ Projects.

Usage

```
createConQuestProject(prefix = getwd(), ...)
```

Arguments

`prefix`	a valid file path where to create project folders.
`...`	optional params, including "setDebug"

Value

Boolean TRUE.

Examples

```
## Not run:
createConQuestProject()

## End(Not run)
```

`DecompressSys` *DecompressSys*

Description

Internal function to decompress an ’ACER ConQuest’ system file that has been compressed using zlib.

Usage

```
DecompressSys(myFile)
```

Arguments

`myFile`	An connection to a compressed ’ACER ConQuest’ system file created by the `put` command in ’ACER ConQuest’.‘

Value

A connection to an uncompressed system file in the users temp dir.

See Also

conquestr::ConQuestSys()

`fisherTrnsfrm` *fisherTrnsfrm*

Description

Helper function to apply Fisher's transformation to a correlation matrix.

Usage

```
fisherTrnsfrm(myCorMat)
```

Arguments

`myCorMat`	A correlation matrix.

Value

A correlation matrix with Fisher's transform applied to values -1 > x > 1.

`fmtCqItanal` *fmtCqItanal*

Description

helper function to produce nicely formatted summary tables from a ConQuest Itanal.

Usage

```
fmtCqItanal(
  cqItanal,
  groups = "all",
  itemNumber = "all",
  ptBisFlag = 0,
  textColHighlight = "red",
  valueDecPlace = 2
)
```

Arguments

`cqItanal`	An ACER ConQuest itanal list object returned by function `getCqItanal`.
`groups`	a vector of group levels to include in the format.
`itemNumber`	a vector of generalised item numbers to format.
`ptBisFlag`	Something.
`textColHighlight`	Something.
`valueDecPlace`	Something.

Value

A list

Examples

```
myEx1Sys <- ConQuestSys()
myEx1Sys_itanal <- getCqItanal(myEx1Sys)
myItanalSummary <- fmtCqItanal(myEx1Sys_itanal)
print(myItanalSummary[[1]])
```

`genItems` *genItems*

Description

Generates a list of item parameter matrices for use in function like `conquestr::genResponses` and `conquestr::informationWrightMap`

Usage

```
genItems(n, scores = NULL, deltadots, taus, discrims = 1, centre = NULL)
```

Arguments

`n`	How many items?
`scores`	When NULL it is assumed that all items have integer scoring, increasing for each category k, and beginning from 0. Otherwise a list where the elements are, in order: • a string naming a distribution function (for example `runif`, `rnorm`) to generate random deviates from (the scores). • a list of parameters to pass to the distribution function (for example, for `runif`, a list of length 2 defining "min" and "max"). This list is assumed to be in order to be directly passed into the function. • a boolean indicating whether the scores should be forced to be increasing across the response categories.

	• optionally a vector of item numbers to apply scores too. If not provided it is assumed that all items will be scored.
`deltadots`	A list that describes the sampling distribution from which item location paramters are drawn. The elements of the list are, in order: • a string naming a distribution function (for example `runif`, `rnorm`) to generate random deviates from (the delta dots). • a list of parameters to pass to the distribution function (for example, for `runif`, a list of length 2 defining "min" and "max"). This list is assumed to be in order to be directly passed into the function. If the argument is missing, item location distribution is assumed to be ~U(-2, 2).
`taus`	A list that describes the sampling distribution from which taus are drawn. Taus are deviations away from the average item location parameter. The elements of the list are, in order: • a string naming a distribution function (for example `runif`, `rnorm`) to generate random deviates from (the taus). • a list of parameters to pass to the distribution function (for example, for `runif`, a list of length 2 defining "min" and "max"). This list is assumed to be in order to be directly passed into the function. • a Boolean indicating whether the taus should be forced to be increasing across the response category boundaries (that is, no items exhibit disordered thresholds). • optionally, a vector that describes the number of response categories to apply to each of the `n` items being sampled. The length of this vector must be equal to `n`. For example if `n`=10, and the first 5 items are polytomous, then a vector of length 10, e.g., c(3, 3, 3, 4, 5, 2, 2, 2, 2, 2). In this example, the first three items have 3 categories each, the fourth item has 4 categories, the fifth item 5 categories, and the last five items are all dichotomies. When missing, a random vector is generated from `sample(c(2:5), 10, replace = TRUE, prob = c(0.4, 0.3, 0.2, 0.1))` to create a mix of dichotomous and polytomous items. When the argument is missing, all items are assumed to be dichotomies.
`discrims`	A list that describes the sampling distribution from which discrimination parameters are drawn. • a string naming a distribution function (for example `runif`, `rnorm`) to generate random deviates from (the discriminations). • a list of parameters to pass to the distribution function (for example, for `runif`, a list of length 2 defining "min" and "max"). This list is assumed to be in order to be directly passed into the function. • a Boolean indicating whether the discriminations are forced to be positive. • a Boolean indicating whether the discrimination is constant within item. When FALSE a unique discrimination is sampled for each category (that is, this can be one way of specifying the Bock Nominal model). • optionally a vector of item indices to sample a discrimination for. The length of this vector must be equal to or less than `n`. For example if `n`=10, and the user wants to sample discriminations for items 1, 5, and 10 then the vector is c(1, 5, 10). All other items will have a discrimination of 1.

	Otherwise it is assumed that all items have sampled discriminations. When missing all items are assumed to have constant discrimination equal to 1.
`centre`	A number indicating the value to centre the generated values in deltadots on. Typically 0 for identification purposes. If NULL then values are left at their generated values (e.g., deviating from the expected mean proportional to sampling error).

Value

A list of item matrices.

See Also

`simplef()`, `genResponses()`, `browseVignettes("conquestr")`

Examples

```
myItem <- matrix(c(0, 0, 0, 0, 1, 1, 0, 1), ncol = 4, byrow = TRUE)
myItems <- list(myItem, myItem)
myItems[[2]][2, 2] <- -1 # make the second item delta equal to -1
myResponses <- genResponses(abilities = rnorm(100), itemParams = myItems)
```

`genResponses` *genResponses*

Description

Generates response vectors for `n` cases to `i` items given known item parameters, person abilities, and (optionally) other inputs.

Usage

```
genResponses(
  abilities,
  groups = NULL,
  itemParams,
  BMatrix = 1,
  mcarP = 0,
  perturbR = NULL
)
```

Arguments

`abilities`	A person by latent-dimension matrix of abilities. One column per dimension.
`groups`	A vector of factors of the same length as `abilities` that allocates each case to a group. Used in perturbR. Defaults to NULL such that all cases are in the one group.

`itemParams`	A list of item parameters of the structure used in `simplef` (a matrix of k categories by four (category score, delta dot, tau, discrimination)). See conquestr::makeItemList for a helper to generate this list.
`BMatrix`	A simplified B-matrix mapping dimensions (columns) to items (rows). Or the integer "1" if items are dichotomous and ability is uni-dimensional.
`mcarP`	A double indicating the proportion of missing data under the MCAR assumption.
`perturbR`	A list of lists, where each element of the list refers to one item and contains a list of elements describing how responses to that item should be perturbed to model misfit. Each element of the list should contain, in order: • item number (int). Which item in `itemParams` is affected, • type of perturbation (string) to apply. One of – "discrimination" - increases or decreases the discrimination of the item at a location specified by the user. – "shift" - increases or decreases the location of the item as to create a uniform shift in the CCC. – ...more to come, • scoring perturbation factor (double). When the type is "discrimination", this defines the scale that the discrimination is increased or decreased. For example, if the item has discrimination of 1, and the perturbation factor is 1.2, the resulting probabilities will be calculated assuming the discrimination is 1 * 1.2 = 1.2. Note that is the value given here is 1, than this kind of perturbation is the same as "shift". When the type is "shift" this value is always ignored. • pivot point (double), When the type is "discrimination", this defines the location around which the perturbation is applied relative to the delta dot. That is, when the type is "discrimination" and the "perturbation factor" is > 1, probabilities above the pivot point will be overestimated (generated responses will higher than expectation) and probabilities below the pivot point will be underestimated (generated responses will lower than expectation). When the pivot point is 0, this calculation happens at the item location parameter (e.g., at the category boundary). When the type is "shift", this is the value added to the item location (delta dot) as to create a uniform shift (DIF) for the group. • group (string). The group found in `groups that should be perturbed. note that if groupsis n` than this value is ignored and all cases' responses are perturbed.

Value

A matrix, `n` cases by `i` items, of scored item responses.

See Also

`simplef()`, `browseVignettes("conquestr")`

Examples

```
myItem <- matrix(c(0, 0, 0, 0, 1, 1, 0, 1), ncol = 4, byrow = TRUE)
myItems <- list(myItem, myItem)
myItems[[2]][2, 2] <- -1 # make the second item delta equal to -1
myResponses <- genResponses(abilities = rnorm(100), itemParams = myItems)
```

`getCqChain` *getCqChain*

Description

creates a data frame representation of the estimation chain from an MCMC model. The burn is discarded and only the unskipped itterations in MCMC chain are retained.

Usage

```
getCqChain(myCqs)
```

Arguments

`myCqs`	A system file.

Value

A data frame.

Examples

```
## Not run:
getCqChain(ConQuestSys())

## End(Not run)
```

`getCqData` *getCqData*

Description

Get data objects from an R object of class ConQuestSys. This function returns person IDs, response data, case estimates, regression and weight data. Each data type is stored as a data frame, and each data frame is a named element of a list.

1. PID,
2. Responses,
3. Estimates,
4. Regression.

Usage

```
getCqData(mySys)
```

Arguments

`mySys`	An R object of class ConQuestSys, returned by the function conquestr::ConQuestSys

Value

A List of data frames.

See Also

conquestr::ConQuestSys()

Examples

```
mySys <- ConQuestSys()
myData <- getCqData(mySys)
```

`getCqDataDf` *getCqDataDf*

Description

Takes a list object returned by `conquestr::getCqData` and coerces it to a wide data frame. This can sometimes cause issues in complex data, for example where there are multiple response vectors for each case (for example a many-facets model). This is because it is assumed that the data can be reduced to a matrix of *gNCases x m variables* (where *m* is the number of id, item, estimate and regression variables in the analysis). For more complex data, the user should use the outputs of `conquestr::getCqData` to manually merge together a data frame.

Usage

```
getCqDataDf(cqData)
```

Arguments

`cqData`	An R object of class list, returned by the function conquestr::getCqData

Value

A data frame containing R data frames based on the list objects in the ConQuest system file that has been read in.

See Also

conquestr::ConQuestSys()

conquestr::getCqData

Examples

```
mySys <- ConQuestSys()
myData <- getCqData(mySys)
suppressWarnings(myDataDf <- getCqDataDf(myData)) # NAs introduced by coercion
```

`getCqFit` *getCqFit*

Description

creates a data frame representation of the fit of parameters in the item reponse model

Usage

```
getCqFit(myCqs)
```

Arguments

`myCqs`	A system file.

Value

A data frame.

Examples

```
## Not run:
getCqFit(ConQuestSys())

## End(Not run)
```

`getCqHist` *getCqHist*

Description

creates a data frame representation of the iteration history for all parameters.

Usage

```
getCqHist(myCqs, labelParams = FALSE)
```

Arguments

`myCqs`	An ACER ConQuest system file created using the conquest command, put.
`labelParams`	A boolean. When true, and if long (user) parameter labels are available, replace default history column names (e.g., "Xsi1") with user labels (e.g., "Item one"). Currently only available for Xsi and Tau.

Value

A data frame.

See Also

`getCqChain()` which is a wrapper for this function to use with models estimated by Markov chain Monte Carlo (MCMC) methods.

Examples

```
myHist <- getCqHist(ConQuestSys(), labelParams = TRUE)
str(myHist)
```

`getCqItanal` *getCqItanal*

Description

helper function to return list of lists, each list relates to one generalised item from an ACER ConQuest `itanal` output. Each list contains: (1) item-total and item-rest correlations ....

Usage

```
getCqItanal(sysFile, matrixPrefix = NULL, isDebug = FALSE)
```

Arguments

`sysFile`	An ACER ConQuest system file.
`matrixPrefix`	The name of the itanal analysis defined in ACER ConQuest. For example, in ACER ConQuest syntax `itanal ! matrixout = itan, ... ;` the value of `matrixPrefix` is "itan". Note, this is required as an ACER ConQuest system file can contain outputs from several calls to *itanal*. A common use, for example, is to call *itanal* for an overall analysis, and a second call to *itanal* for group-level analysis.
`isDebug`	A boolean to toggle on or off debug output

Value

A list.

Examples

```
myItanal <- getCqItanal()
print(myItanal[[1]])
```

`getCqItanalFacility` *getCqItanalFacility*

Description

returns an item facility for each item in itanal object created by ACER ConQuest. For a dichotomously scored Rasch-like item, facility is the percent correct. For a polytomously scored item, or with estimated scores, facility is given by: the sum of the number of cases in each response category, multiplied by the score for that category divided by the sum of all cases responding to the items times the maximum score for the item.

Usage

```
getCqItanalFacility(itan)
```

Arguments

`itan`	A list of class "cqItanal" created by `conquestr::getCqItanal()`

Value

A list.

Examples

```
mySys <- ConQuestSys()
myItan <- getCqItanal(mySys)
getCqItanalFacility(myItan)
```

`getCqItanalSummary` *getCqItanalSummary*

Description

returns an itanal as a data frame in summary format: one row per generalised item with:

- item label
- valid N
- facility (see `conquestr::getCqItanalFacility`)
- item-rest correlation
- item-total correlation
- fit (infit/weighted MNSQ) if available
- item locations (deltas)

Usage

```
getCqItanalSummary(itan)
```

Arguments

`itan`	A list of class "cqItanal" created by `conquestr::getCqItanal()`

Value

A data frame.

Examples

```
mySys <- ConQuestSys()
myItan <- getCqItanal(mySys)
getCqItanalSummary(myItan)
```

`getCqRespModel` *getCqRespModel*

Description

produces a table of model parameter estimates, errors, fits, and scaled 2PL estimates if available.

Usage

```
getCqRespModel(sysFile)
```

Arguments

`sysFile`	An ACER ConQuest system file read into R using conquestr::ConQuestSys

Value

A List of data frames. Each data frame is a term in the response model

Examples

```
## Not run:
myShowRespMod <- getCqRespModel(conquestr::ConQuestSys())

## End(Not run)
```

`getCqTerms` *getCqTerms*

Description

creates a data frame representation of the terms of the model statement, including interactions.

Usage

```
getCqTerms(myCqs)
```

Arguments

`myCqs`	A system file.

Value

A data frame.

Examples

```
## Not run:
getCqTerms(ConQuestSys())

## End(Not run)
```

`getCqVars` *getCqVars*

Description

creates a data frame representation of the variables in the model statement. Note that steps are not variables.

Usage

```
getCqVars(myCqs)
```

Arguments

`myCqs`	A system file.

Value

A data frame.

Examples

```
## Not run:
getCqVars(ConQuestSys())

## End(Not run)
```

`ginsOnDims` *ginsOnDims*

Description

returns a list of length gNDims. Each element of the list contains a vector of the gins on this dim.

Usage

```
ginsOnDims(sysFile)
```

Arguments

`sysFile`	An ACER ConQuest system file read into R using conquestr::ConQuestSys

Value

a list

Examples

```
## Not run:
myResult <- ginsOnDims(conquestr::ConQuestSys())

## End(Not run)
```

`informationWrightMap` *informationWrightMap*

Description

Plots test information function, relative to ability density, and item locations.

Usage

```
informationWrightMap(
  myItems,
  myAbilities,
  type = "empirical",
  minTheta = NA,
  maxTheta = NA,
  stepTheta = NA,
  scaleInfo = 1,
  plotItemPoints = "deltadots"
)
```

Arguments

`myItems`	A list of matricies describing item parameters.
`myAbilities`	A vector of person abilities on one dimension.
`type`	A character String. Should the test information be calculated empirically ("empirical" - default) or analytically using moments of distribution ("approx").
`minTheta`	The smallest value of ability PDF to plot.
`maxTheta`	The largest value of ability PDF to plot.
`stepTheta`	The increment to iterate over the ability PDF. Defaults to 0.01.
`scaleInfo`	A scaling factor to apply to the plot of tesr information. Because ability distribution is a PDF with area one, and a test information function has area L, this can make the plot more interpretable. Defaults to 1.
`plotItemPoints`	A character string indicating what item points should be plotted along the x-axis. similar to the histogram of item locations plotted on a Wrightmap. Can be "none", "deltadots", "thresholds".

Value

A ggplot2 object.

Examples

```
myDeltaDots <- data.frame(
  id = c(1:10),
  itemid = paste0("item", 1:10),
  delta = rnorm(10)
)
MyTaus <- data.frame(
  id = c(2L, 10L),
  itemId = NA,
  step = c(1L, 1L),
  tau = rnorm(2)
)
myItemList <- makeItemList(deltaDot = myDeltaDots, tau = MyTaus)
myInfoPlot <- informationWrightMap(myItemList, rnorm(1000, 0, 1), minTheta=-5, maxTheta=5)
```

`infoWI` *infoWI*

Description

Calculates an index representing the product of a test information function and an ability distribution.

Usage

```
infoWI(myItems, myAbilities, type = "empirical")
```

Arguments

`myItems`	A vector of item deltas.
`myAbilities`	A vector of person abilities.
`type`	A character String. Should the test information be calculated empirically ("empirical" - default) or analytically using moments of distribution ("approx").

Value

A double.

Examples

```
infoWIOut <- infoWI(runif (10, -2, 3), rnorm(1000, 0, 1))
```

`itemInfoAtTheta` *itemInfoAtTheta*

Description

Calculates item information at a value of theta given a set of item parameters for one item.

Usage

```
itemInfoAtTheta(myItem, theta)
```

Arguments

`myItem`	A matrix of item parameters of the structure used in `simplef`
`theta`	A number.

Examples

```
anItem <-  matrix(c(0,0,0,1,1,1,0,1), nrow = 2, byrow = TRUE)
itemInfoAtTheta(anItem, 0)
```

`itemInfoOverTheta` *itemInfoOverTheta*

Description

Calculates item information over a rage of theta given a set of item parameters. Returns a data frame with item information at a discrete set of values of theta. This is useful for plotting item information functions.

Note this function is redundant - use testInfoOverTheta and pass a single item as a list.

Usage

```
itemInfoOverTheta(myItem, minTheta = -6, maxTheta = 6, stepTheta = 0.1)
```

Arguments

`myItem`	A matrix of item parameters of the structure used in `simplef`
`minTheta`	The smallest value of ability PDF to calculate info and to plot. Defaults to -6.
`maxTheta`	The largest value of ability PDF to calculate info and to plot. Defaults to 6.
`stepTheta`	The increment to iterate over the ability PDF. Defaults to 0.01.

Examples

```
anItem <- matrix(c(0,0,0,1,1,1,0,1), nrow = 2, byrow = TRUE)
itemInfoOverTheta(anItem)
```

`itemListToThresholds` *itemListToThresholds*

Description

Taskes a list of item parameter matricies and returns a data frame containing Thurstonian Thresholds (*gammas*) for all items. Thurstonian thresholds are the location on the trait/scale at which the cumulative probability of being in category k, or any higher category equals some probability (usually 0.5, the default). Thurstonian thresholds are considered a way of describing the difficulty of polytomously scored items and are usually the value used in visualisations like Wright maps. Thurstonian thresholds can only be calculated for items where response categories are scored such that each category can be placed in an order increasing scores (e.g., no ties as per the Ordered Partition model)

Usage

```
itemListToThresholds(
  myItems,
  threshP = 0.5,
  minTheta = -20,
  maxTheta = 20,
  convC = 1e-05
)
```

Arguments

`myItems`	A list of item parameter matricies of the structure used in `simplef` (a matrix of k categories by four (category score, delta dot, tau, discrimination)).
`threshP`	The probability at which the thresholds are calculated (defaults to the usual value of 0.5)
`minTheta`	The lower-bound starting value of the split-half search used to find the threshold for the category.
`maxTheta`	The upper-bound starting value of the split-half search used to find the threshold for the category.
`convC`	The convergence criteria used to determine when the threshold has been found. The difference between `threshP` and the cumulative probability of the category and any higher category at the current value of theta (the current proposed value of threshold being tested).

Value

A data frame including 4 columns:

- id, an integer index reflecting which item this is, in the same order as myItems
- itemid, a string with the names from the items in myItems (NA if item list is not named)
- step, which step does this threshold belong?
- location, the value of the threshold

Examples

```
myItem <- matrix(
  c(
    0, -0.58    ,  0     , 1,  # delta+tau   thurst thresh (gamma)
    1, -0.58    ,  0.776 , 1,  # 0.196       -1.14
    2, -0.58    , -0.697 , 1,  # -1.277      -0.93
    3, -0.58    , -0.629 , 1,  # -1.209      -0.64
    4, -0.58    ,  0.55  , 1   # -0.03        0.25
    ), ncol =4, byrow=TRUE
 )
itemListToThresholds(list(myItem))
```

`makeItemDfs` *makeItemDfs*

Description

takes in a list of item matrices and returns a list of data frames each representing the parameters given in the matrices. The return object is suitable to pass into `conquestr::makeItemList` to construct a lit of matrices where each matrix represent one item's set of item parameters. The structure of the matrix is the same as used in `conquestr::simplef` (a matrix of k categories by four (category score, delta dot, tau, discrimination)). A common use for this function is turn a list of item matrices into a flat data structure.

Usage

```
makeItemDfs(itemList)
```

Arguments

`itemList`	a list of item matrices. The structure of each matrix is the same as used in `conquestr::simplef` (a matrix of k categories by four (category score, delta dot, tau, discrimination)).

Value

a list.

Examples

```
nItems <- 10
myItemsDeltaDot <- data.frame(
  id= seq(nItems),
  itemid= NA,
  delta = runif (nItems, -4, 1) # nItems items in range -4,1
)
myItemsList <- conquestr::makeItemList(deltaDot = myItemsDeltaDot)
```

`makeItemList` *makeItemList*

Description

creates a list of item matrices. Each matrix represent one item's set of item parameters. The structure of the matrix is the same as used in `conquestr::simplef` (a matrix of k categories by four (category score, delta dot, tau, discrimination)).

Usage

```
makeItemList(scores = NULL, deltaDot, tau = NULL, discrim = 1)
```

Arguments

`scores`	a data frame or matrix containing category scores for each item. If NULL, it is assumed increasing integer scoring starting at 0 is used for all items (that is, the first category is scored 0, the second category is scored 1, the k^{th} category is scored k-1). If a data frame, column labels should be "id", "itemid", "step", "score". If a matrix, the column order should be: "id", a unique item ID for each item matched with values in `deltaDot`; "itemid", item labels for each item (or NA); "step", an indicator of which step/item category this score represents and "score" the value for the scoring parameter associated with this category. There must be one score for each category (i.e. 2 for dichotomies and one for each of k categories for polytomies). If a data frame, or a matrix: • "id" is an integer • "itemid" is a character string • "step" is an integer • "score" is numeric • The original category scores (i.e., increasing integer scoring) is preserved in the rownames of the matrix.
`deltaDot`	a data frame or matrix of delta dots (average item location/difficulty for each item). If a data frame, column labels should be: "id", "itemid", "delta". "itemid" should be populated with an item label or be missing for all values. If a matrix, column order should be: "id", a unique item ID for each row; "itemid", item labels for each item (or NA); "delta", a delta dot. If a data frame, or a matrix: • "id" is an integer • "itemid" is a character string • "delta" is numeric
`tau`	NULL if all items are dichotomies. A data frame or matrix of taus for polytomous items. Only polytomous items should be in this file. If an item ID in `deltaDot` in not in `tau` it is assumed that the item is dichotomous. The tau parameters represent the deviation from the delta dot to give the item parameters for adjacent category boundaries (e.g., delta one ($\delta_1 = \dot{\delta} + \tau_1$) is the boundary between k_1 and k_2, delta two ($\delta_2 = \dot{\delta} + \tau_2$) is the category boundary between k_1 and k_2). Where a polytomous item has k categories, there should be k-2 rows for that item in `tau`. For example, a 3-category item has categories k_1, k_2 and k_3. There will be one value in `tau` for this item. The value in `tau` represents the the first category boundary. (e.g., between k_1 and k_2). The last (second in this case) category boundary is constrained to be the negative sum of the other tau values within this item (and is therefore not required in the file).

If a data frame, column labels should be "id", "itemid", "step", "tau". If a matrix, the column order should be: "id", a unique item ID for each item matched with values in `deltaDot`; "itemid", item labels for each item (or NA); "step", an indicator of which step/item category this threshold represents (minimum value should be 1 and maximum value should be k-1); "tau" the value for the tau parameter associated with this step.

If a data frame, or a matrix:

- "id" is an integer
- "itemid" is a character string
- "step" is an integer
- "tau" is numeric

`discrim`	a double, a data frame, or a matrix of item (or category) discrimination parameters. When a double is provided, the value is applied to all discrimination parameters. The default is 1. Setting the value to 1.7 is one approach to re-scale to the normal ogive metric. Otherwise a data.frame or matrix defining the discrimination parameter for each response category. If a data frame, column labels should be "id", "itemid", "step", "discrim". If step is NA and there is only one entry for an item "itemid", the discrimination is assumed to be constant for all response categories with the item. This is the case for named models like the GPCM and 2PL models, and can be a short hand way of defining the discrimination without specifying all categories. When discrimination varies across scoring categories, the Bock-nominal model is implied. In the case of discrimination varying across scoring categories, all categories must be defined.

If a data frame, or a matrix:

- "id" is an integer
- "itemid" is a character string
- "step" is an integer
- "discrim" is numeric

Value

a list.

Examples

```
nItems <- 10
myItemsDeltaDot <- data.frame(
  id= seq(nItems),
  itemid= NA,
  delta = runif (nItems, -4, 1) # nItems items in range -4,1
)
myItemsList <- conquestr::makeItemList(deltaDot = myItemsDeltaDot)
```

`plotCCC` *plotCCC*

Description

Creates a plot of an item characteristic curve (by response category). For a dichotomous item, this will yield a single curve, for polytomous items this will produce a curve for each response category. Note this is not for use with `rout` files. See the generic function `plotRout` for plotting rout files.

Usage

```
plotCCC(
  item,
  abilities,
  responses,
  weights = NULL,
  groups = NULL,
  range = c(-6, 6),
  by = 0.1,
  linetype = "bins",
  bins = 10,
  plotZero
)
```

Arguments

`item`	A matrix of item parameters for a single item. Matrix should be of the form used in `simplef`
`abilities`	A vector of doubles estimated person abilities.
`responses`	A vector of integers giving the observed person responses to this item.
`weights`	A vector of doubles of sampling weights.
`groups`	A factor vector indicating groups.
`range`	Lower and upper bounds to plot over (defaults to `c(-6, 6)`).
`by`	A double. The increment to the sequence along `range` used to plot the model lines.
`linetype`	A string. Should the empirical lines be based on "bins", or "regression". Defaults to "bins"
`bins`	If *linetype* is "bins", how many bins should be used to chunk the empirical lines? defaults to 10. Ignored otherwise.
`plotZero`	Should the zero category be plotted? Defaults to `FALSE` when item is dichotomous and `TRUE` otherwise.

Value

A ggplot2 object.

Examples

```
myRout <- ConQuestRout()
myPlot <- plotRout(myRout)
## Not run:
# if you run the above example you will have an ICC plot in the object `myPlot`.
plot(myPlot)

## End(Not run)
```

---

plotCqHist *plotCqHist*

Description

generates a plot from a history object. Use `getCqHist` to create a history object from an 'ACER ConQuest' system file.

Usage

```
plotCqHist(
  myHist,
  centre = TRUE,
  params = c("all"),
  legend = FALSE,
  plotProblems = NULL
)
```

Arguments

`myHist`	an R object created by the `getCqHist` function.
`centre`	a Boolean representing whether the iteration history should be mean centred (within parameter). This is helpful for plots that include all parameters to ensure the Y axis is sensible. Consider, for example, the readability of a plot with raw values of the Likelihood *and* item parameters on it.
`params`	A string of which params to plot. Must be one or more of "all", "Likelihood", "Beta", Variance", "Xsi", "Tau". Note the match when using "Beta", Variance", "Xsi", "Tau" is by regular expression, so "Xsi1" will plot item location parameter 1, 10-19, 100-199 and so on.
`legend`	Should a legend be plotted?
`plotProblems`	an optional list defining which potential problem parameters to plot. • Iters: The first element of the list is an integer defining how many of the final iterations to consider (e.g., identify parameters that are moving the most over the final 20 iterations). if NA, the default is to consider the last 10% of iterations.

- Magnitude: The second element of the list is number indicating the magnitude of change over the last n iterations. if NA, and *Type* is "relative", defaults to 30 times the largest change at the final iteration. if NA, and *Type* is "absolute", defaults to 0.05 logits.
- Type: The third element of the list is a string, either "relative" or "absolute":
  - "relative" indicates that *Magnitude* is the multiple of the change between the final iteration and the second-to-last iteration that indicates a potential problem.
  - "absolute" indicates that *Magnitude* refers to change between the the final iteration and the value in *Iters* that indicates a potential problem.

Value

A ggplot2 object.

Examples

```
## Not run:
myHistPlot <- plotCqHist(getCqHist(ConQuestSys()))

## End(Not run)
```

`plotDif` *plotDif*

Description

Creates a plot (ggplot2 object) of item parameter estimates common to two system files (e.g., a DIF analysis).

Usage

```
plotDif(mySysToItemDifDf, myScale = "centred", mySuffixes)
```

Arguments

`mySysToItemDifDf`	An R object of class data frame returned from conquestr::sysToItemDifDf
`myScale`	A string specifying if the item parameter estimates displayed should be "centred" (default), "scaled" (z scores), or "none" (raw).
`mySuffixes`	a vector of strings specifying the names for the two groups being analysed, e.g., if the two system files are an analysis of boys and girls, the vector may be `c(_male", "_female")`.

Value

A ggplot2 object.

See Also

conquestr::sysToItemDifDf()

Examples

```
mySys1 <- ConQuestSys()
mySys2 <- ConQuestSys()
mySysList <- list(mySys1, mySys2)
myDifDf <- sysToItemDifDf(mySysList, mySuffixes = c("_male", "_female"), myDims = "all")
myDifPlot <- plotDif (myDifDf,myScale = "centred", mySuffixes = c("_male", "_female"))
## Not run:
# if you run the above example you will have the plot in the object `myDifPlot`.
plot(myDifPlot)

## End(Not run)
```

`plotItemMap` *plotItemMap*

Description

Creates a plot (ggplot2 object) of item parameter estimates and abilities on latent trait. Note this is not for use with `rout` files. See the method method plotRout.itemMap to the generic function `plotRout`

Usage

```
plotItemMap(mySys, myDims = "D1", ginLabs = "short", abilityType = "PV", ...)
```

Arguments

`mySys`	An 'ACER ConQuest' system file object created using the conquestr::ConQuestSys function.
`myDims`	A string specifying which specific dimensions should be included. The default is "D1", Specific dimensions are specified by the label "D1" for dimensions 1 etc.
`ginLabs`	A string specifying whether short or long gin labels should be used. Default to "short".
`abilityType`	What kind of person ability estimate should be used? Defaults to plausible values. Alternatively WLE, MLE, EAP.
`...`	Optional arguments, mostly for debugging, e.g., `setDebug = TRUE` will print temporary data frames.

Value

A ggplot2 object.

Examples

```
mySys1 <- ConQuestSys()
myItemMap <- plotItemMap(mySys1)
## Not run:
# if you run the above example you will have the plot in the object `myItemMap`.
plot(myItemMap)

## End(Not run)
```

---

plotModelCCC *plotModelCCC*

Description

Creates a plot of a model implied category characteristic curve. Note this is not for use with `rout` files. See the generic function `plotRout` for plotting rout files.

Usage

```
plotModelCCC(item, range = c(-6, 6), by = 0.1, plotZero)
```

Arguments

`item`	Item parameters for a single item.
`range`	Lower and upper bounds to plot over (defaults to c(-6, 6).
`by`	Increment to the sequence along ‘range“.
`plotZero`	Should the zero category be plotted? Defaults to `FALSE` when item is dichotomous and `TRUE` otherwise.

Value

A ggplot2 object.

Examples

```
myItem <- matrix(
  c(
    0, 0, 0, 1,
    1, 1, 0, 1
  ),
  ncol = 4, byrow=TRUE
)
myPlot <- plotModelCCC(myItem)
```

`plotModelExp` *plotExpected*

Description

Creates a plot of an item- or test- expected score curve. If ability estimates are provided, both empirical and model curves are produced. Can optionally handle weights and groups as required. Note this is not for use with `rout` files. See the generic function `plotRout` for plotting rout files.

Usage

```
plotModelExp(
  items,
  range = c(-6, 6),
  by = 0.1,
  bins = NULL,
  abilities = NULL,
  weights = NULL,
  group = NULL,
  scale = FALSE
)
```

Arguments

`items`	a *list* of one or more matrices of item parameters. Used in producing model-implied curves.
`range`	Lower and upper bounds to plot over (defaults to c(-6, 6). Used in producing model-implied curves. For empirical curves a range is chosen given the min and max values in abilities.
`by`	Increment to calculate expectation along `range`. Used in producing model-implied curves.
`bins`	A double. Optional. How many equally sized bins should abilities be broken up into? Used in producing empirical curves. If not provided and abilities are provided, a suitable value is chosen given the length of abilities.
`abilities`	A vector of doubles. Optional.
`weights`	A vector of doubles. Optional.
`group`	A vector of type factor. Optional.
`scale`	A Boolean. Whether plot should be scaled such that the Y-axis ranges from 0 to 1.

Value

A ggplot2 object.

Examples

```
myItem <- matrix(
  c(
    0, 0, 0, 1,
    1, 1, 0, 1
  ),
  ncol = 4, byrow=TRUE
)
myPlot <- plotModelExp(list(myItem))
```

`plotRout` *plotRout*

Description

generates a plot from an 'ACER ConQuest' Rout file. use `ConQuestRout` to read in an Rout file created by a `plot` command in 'ACER ConQuest'.

Usage

```
plotRout(myRout, ...)

## S3 method for class 'TestInfo'
plotRout(myRout, ...)

## S3 method for class 'InformationWithLatentDist'
plotRout(myRout, ...)

## S3 method for class 'ICC'
plotRout(myRout, ...)

## S3 method for class 'MCC'
plotRout(myRout, ...)

## S3 method for class 'TCC'
plotRout(myRout, ...)

## Default S3 method:
plotRout(myRout, ...)
```

Arguments

`myRout`	an R object created by the `ConQuestRout` function.
`...`	additional arguments passed into plotting functions

Value

A ggplot2 object.

Examples

```
myRout <- ConQuestRout()
myPlot <- plotRout(myRout)
## to see why we import this, see https://ggplot2.tidyverse.org/articles/ggplot2-in-packages.html
```

`pvMeanVar` *pvMeanVar*

Description

Applies the law of total variance (EVEs law) to calculate the mean and variance of a set of PVs for one dimension.

Usage

```
pvMeanVar(myData)
```

Arguments

`myData`	A matrix of PVs for one dimension: m PVs by n cases.

Value

A list containing the mean and variance of the PVs.

`q3ExpCorrect` *q3ExpCorrect*

Description

Helper function to apply correction to correlation matrix. When working with standardised residuals, the expectation of the correlations is -1/(L-1) rather than 0 See DOI: 10.1177/0013164410379322

Usage

```
q3ExpCorrect(myCorMat)
```

Arguments

`myCorMat`	A correlation matrix.

Value

A correlation matrix with the Q3 statistic correction applied.

`ReadSys` *ReadSys*

Description

Internal function to read an 'ACER ConQuest' system file. Called by conquestr::ConQuestSys.

Usage

```
ReadSys(myFile, isMini)
```

Arguments

`myFile`	An connection to an 'ACER ConQuest' system file created by the `put` command in 'ACER ConQuest'. If the file is compressed, and uncompressed temporary file is created.
`isMini`	A boolean, set to TRUE if the system file is a *mini* system file created by 'ACER ConQuest' command put with option "mini = yes".

Value

A list containing the data objects created by 'ACER ConQuest'.

See Also

conquestr::ConQuestSys()

`ReadSysMini` *ReadSysMini*

Description

Internal function to read an 'ACER ConQuest' system file. Called by conquestr::ConQuestSys.

Usage

```
ReadSysMini(myFile, Dimensions, N, NPlausibles, isDebug)
```

Arguments

`myFile`	An 'ACER ConQuest' *mini* system file created by the `put` command in 'ACER ConQuest' with the option "mini = yes". The put command must use the option `compressed = no`.
`Dimensions`	.
`N`	.
`NPlausibles`	.
`isDebug`	.

Value

A list containing the data objects created by 'ACER ConQuest'.

See Also

conquestr::ConQuestSys()

`recodeResps` *recodeResps*

Description

Recode raw item responses for analyses.

Usage

```
recodeResps(data, recodeMap, varLabel, rawLabel, recodeLabel)
```

Arguments

`data`	Raw data, a data frame.
`recodeMap`	A data frame which contains the raw responses and corresponding recoded responses for each of the items in long form.
`varLabel`	A variable name in recodeMap that identifes the item label.
`rawLabel`	A variable name in recodeMap that identifies the raw item responses to be recoded.
`recodeLabel`	A variable name in recodeMap that idenitfies the new values to recode to.

Value

a data frame with raw data recoded according to recodeMap.

`replaceInDataFrame` *iterate through a data frame and use replaceInVector*

Description

iterate through a data frame and use replaceInVector

Usage

```
replaceInDataFrame(d, r, x)
```

Arguments

`d`	A DataFrame.
`r`	A double - the value to be replaced if it is < -1e300.
`x`	A double - the value to replace r with.

`replaceInVector` *replace a very large neagtive number with something - usually NA_REAL*

Description

replace a very large neagtive number with something - usually NA_REAL

Usage

```
replaceInVector(v, r, x)
```

Arguments

`v`	A NumericVector.
`r`	A double - the value to be replaced if it is < -1e300.
`x`	A double - the value to repalce r with.

`searchConQuestSys` *searchConQuestSys*

Description

Search for object names within a ConQuest System file object.

Usage

```
searchConQuestSys(searchString, mySys, value = TRUE, ignore.case = TRUE)
```

Arguments

`searchString`	A string to search within the names of mySys.
`mySys`	An 'ACER ConQuest' system file object created using the conquestr::ConQuestSys function.
`value`	Should searchConQuestSys return the name of the object or its index.
`ignore.case`	Should searchConQuestSys ignore the case of the search term.

Value

a string including object names mathching the search term

`steigerStat` *steigerStat*

Description

Function to cacluate the Steiger statistic. The Steiger statistic is a test of independance of the standardised residuals ((O-E)/sqrt(Var(E))), where Var(E) = p(x)/(1-p(x)).

Usage

```
steigerStat(myDat, q3Adj = TRUE, fisher = TRUE, dfAdj = FALSE, tpm)
```

Arguments

`myDat`	A data frame or matrix containing standardised residuals.
`q3Adj`	A bool indicating whether the Q3 correction should be applied.
`fisher`	A bool indicating whether the Fisher Transform should be applied.
`dfAdj`	A bool indicating whether the df should be adjusted for sample size, L, and targeting. If dfAdj is TRUE, then you must pass in the optional argument `tmp` (test-person match)
`tpm`	A number indicating the test-person match, where 0 indicates that mean item difficulty is equal to mean person ability, and -1 indcates that mean item difficulty is 1 logit below mean person ability.

Value

A list of class "steigerStat" with the Steiger Statistic, correlation matrix, and chi square test.

`summariseCqChain` *summariseCqChain*

Description

takes a data frame created by getCqChain and returns a list reporting the mean and variaince for each parameter

Usage

```
summariseCqChain(myChain)
```

Arguments

`myChain`	A data frame returned from getCqChain.

Value

A list.

Examples

```
## Not run:
summariseCqChain(getCqChain(ConQuestSys()))

## End(Not run)
```

`sysFileOk` *sysFileOk*

Description

checks

Usage

```
sysFileOk(sysFile, defaultSys)
```

Arguments

`sysFile`	An ACER ConQuest system file read into R using conquestr::ConQuestSys
`defaultSys`	A Boolean indicating if sysFile is the defualt system file created by an empty call to conquestr::ConQuestSys

Examples

```
## Not run:
sysFileOkResult <- sysFileOk(conquestr::ConQuestSys())

## End(Not run)
```

`sysToBMatrixDf` *sysToBMatrixDf*

Description

Read an R object of class ConQuestSys and create a labelled representation of the B matrix (scoring matrix). This maps item response categories to items and dimensions. Returns long data frame, where items are duplicated if they are in many dimensions.

Usage

```
sysToBMatrixDf(mySys, applyLabels = TRUE)
```

Arguments

`mySys`	An R object of class ConQuestSys, returned by the function conquestr::ConQuestSys
`applyLabels`	A bool indicating whether labels (e.g., dimension labels) should be appended.

Value

A data frame containing R the labelled B matrix.

Examples

```
myBMatrix <- sysToBMatrixDf(ConQuestSys())
## Not run:
# if you run the above example you will have the B Matrix from the example system file.
str(myBMatrix)

## End(Not run)
```

`sysToItemDifDf` *sysToItemDifDf*

Description

Creates a data frame that includes the common item parameter estimates from two (or more) system files (e.g., a DIF analysis).

Usage

```
sysToItemDifDf(listOfSysFiles, mySuffixes, myDims = "all")
```

Arguments

`listOfSysFiles`	A list of system files returned from conquestr::ConQuestSys
`mySuffixes`	a vector of strings specifying the names for the two groups being analysed, e.g., if the two system files are an analysis of boys and girls, the vector may be `c(_male", "_female")`.
`myDims`	A string specifying if all or specific dimensions should be included. The default is "all", Specific dimensions are specified by the label "D1" for dimensions 1 etc.

Value

A data frame object.

See Also

conquestr::plotDif ()

`testInfoAtTheta` *testInfoAtTheta*

Description

Calculates test information at a value of theta given a list of matricies of item parameters for one or more items.

Usage

```
testInfoAtTheta(myItems, theta)
```

Arguments

`myItems`	A list of matrices of item parameters of the structure used in `simplef`
`theta`	a number.

Examples

```
anItem <- matrix(c(0,0,0,1,1,1,0,1), nrow = 2, byrow = TRUE)
testInfoAtTheta(list(anItem), 0)
```

`testInfoOverTheta` *testInfoOverTheta*

Description

Calculates test information over a range of theta given a list of matricies of item parameters for one or more items. Returns a data frame with item information at a discrete set of values of theta. This is useful for plotting test information functions.

Usage

```
testInfoOverTheta(myItems, minTheta = -6, maxTheta = 6, stepTheta = 0.1)
```

Arguments

`myItems`	a list of item parameters of the structure used in `simplef`
`minTheta`	The smallest value of ability PDF to calculate info and to plot. Defaults to -6.
`maxTheta`	The largest value of ability PDF to calculate info and to plot. Defaults to 6.
`stepTheta`	The increment to iterate over the ability PDF. Defaults to 0.01.

Examples

```
anItem <- matrix(c(0,0,0,1,1,1,0,1), nrow = 2, byrow = TRUE)
testInfoOverTheta(list(anItem))
```

`thrstThrsh` *thrstThrsh*

Description

Generates Thurstonian Thresholds (sometimes called *gammas*) to an item. Thurstonian thresholds are the location on the trait/scale at which the cumulative probability of being in category k, or any higher category equals some probability (usually 0.5, the default). Thurstonian thresholds are considered a way of describing the difficulty of polytomously scored items and are usually the value used in visualisations like Wright maps. Thurstonian thresholds can only be calculated for items where response categories are scored such that each category can be placed in an order increasing scores (e.g., no ties as per the Ordered Partition model)

Usage

```
thrstThrsh(myItem, threshP = 0.5, minTheta = -20, maxTheta = 20, convC = 1e-05)
```

Arguments

`myItem`	A matrix of parameters for a single item of the structure used in `simplef` (a matrix of k categories by four (category score, delta dot, tau, discrimination)).
`threshP`	The probability at which the threshold is calculated (defaults to the usual value of 0.5)
`minTheta`	The lower-bound starting value of the split-half search used to find the threshold for the category.
`maxTheta`	The upper-bound starting value of the split-half search used to find the threshold for the category.
`convC`	The convergence criteria used to determine when the threshold has been found. The difference between `threshP` and the cumulative probability of the category and any higher category at the current value of theta (the current threshold being tested).

Value

A k-1 by 1 matrix with Thurstonian thresholds for this item. Values are NA when the threshold cannot be calculated.

Examples

```
myItem <- matrix(
  c(
    0, -0.58    ,  0     , 1,  # delta+tau   thurst thresh (gamma)
    1, -0.58    ,  0.776 , 1,  # 0.196       -1.14
    2, -0.58    , -0.697 , 1,  # -1.277      -0.93
    3, -0.58    , -0.629 , 1,  # -1.209      -0.64
    4, -0.58    ,  0.55  , 1   # -0.03        0.25
    ), ncol =4, byrow=TRUE
```

```
 )
thrstThrsh(myItem)
```

`transformPvs` *transformPvs*

Description

Helper function to Transform PVs onto a new metric (e.g., PISA Mean = 500, SD = 100). Uses the method described in the PISA 2012 technical manual.

Usage

```
transformPvs(data, mT = 0, sdT = 1, weights = 1)
```

Arguments

`data`	A data frame or matrix that contains the PVs
`mT`	The desired mean of the PVs
`sdT`	The desired sd of the PVs
`weights`	a vector of weights, the same length as `data[1]` used to calculate the mean and SD across the PVs

Value

a List of transformed PVs with as many elements as PVs were listed in 'x'.

`writeImportXsi` *writeImportXsi*

Description

Writes a fixed width text file in the format required for the ACER ConQuest command and argument `import anchor_xsi`. Can also be used for initial values, though caution should be used with the interpretation of the argument `lconstraint` which should relate to the model of interest ACER ConQuest

Currently only works with implicit variables. Explicit variables may be added in the future.

Usage

```
writeImportXsi(items, bmatix = 1L, lconstraint = "none", file)
```

Arguments

`items`	a list of item matrices
`bmatix`	either the integer 1L for a unidimensional model, or a matrix, items by dimensions with 1L representing that the item in on this dimension, and a 0 otherwise.
`lconstraint`	the identification constraint in use, one of "none", "items", or "cases".
`file`	a path and filename to write file to disk. #' @return invisibly returns path of file written to disk)

See Also

`simplef()`, `genResponses()`, `browseVignettes("conquestr")`

Examples

```
myItem <- matrix(c(0, 0, 0, 0, 1, 1, 0, 1), ncol = 4, byrow = TRUE)
myItems <- list(myItem, myItem)
myItems[[2]][2, 2] <- -1 # make the second item delta equal to -1
myResponses <- genResponses(abilities = rnorm(100), itemParams = myItems)
```

Index