Package 'conquestr'

October 26, 2026

Type Package

Title An R Package to Extend 'ACER ConQuest'

Version 1.9.2

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/collections/acer-conquest-5>.

License GPL-3

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

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

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'' ``pvStats.R'' ``residHelpers.R'' ``showHelpers.R'' ``thrstThrsh.R'' ``cleaningHelpers.R''

Depends R (>= 4.1.0)

Config/roxygen2/version 8.1.1

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 2026-10-26 11:50:02 UTC

Contents

checkItemRespValid . . . 3
checkVars . . . 4
cnvrtItemParam . . . 5
ConQuestCall . . . 6
ConQuestRout . . . 7
ConQuestSys . . . 7
createConQuestProject . . . 8
DecompressSys . . . 9
fisherTrnsfrm . . . 9
fmtCqItanal . . . 10
genItems . . . 11
genResponses . . . 13
genResponsesProbs . . . 16
getCqChain . . . 17
getCqData . . . 18
getCqDataDf . . . 18
getCqFit . . . 19
getCqHist . . . 20
getCqItanal . . . 20
getCqItanalFacility . . . 21
getCqItanalSummary . . . 22
getCqRespModel . . . 22
getCqTerms . . . 23
getCqVars . . . 24
ginsOnDims . . . 24
informationWrightMap . . . 25
infoWI . . . 26
itemInfoAtTheta . . . 27
itemInfoOverTheta . . . 27
itemListToThresholds . . . 28
lcs . . . 29
lcs_vec . . . 29
makeItemDfs . . . 30
makeItemList . . . 30
plotCCC . . . 33
plotCqHist . . . 34
plotDif . . . 35

plotItemMap . . . 36
plotModelCCC . . . 37
plotModelExp . . . 38
plotRout . . . 39
pvCor . . . 40
pvCorMatrix . . . 42
pvCovMatrix . . . 44
pvDescribe . . . 46
pvDiff . . . 47
pvIqr . . . 48
pvMean . . . 48
pvMedian . . . 49
pvQuantile . . . 49
pvRegression . . . 50
pvRegression-methods . . . 51
pvSd . . . 52
pvStat . . . 53
pvVar . . . 55
q3ExpCorrect . . . 55
ReadSys . . . 57
ReadSysMini . . . 57
recodeResps . . . 58
replaceInDataFrame . . . 59
replaceInVector . . . 59
searchConQuestSys . . . 60
steigerStat . . . 60
summariseCqChain . . . 63
sysFileOk . . . 63
sysToBMatrixDf . . . 64
sysToItemDifDf . . . 65
testInfoAtTheta . . . 65
testInfoOverTheta . . . 66
thrstThrsh . . . 66
transformPvs . . . 67
writeImportXsi . . . 68

Index **69**

`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", DFrom = 1, DTo = 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)
DFrom	a number, giving the scaling constant for the `from` parameterisation. Default is 1 (logistic metric). Other common values are D = 1.7 (to give the normal ogive metric)
DTo	a number, giving the scaling constant for the `to` parameterisation. 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", DFrom = 1, DTo = 1.7)
```

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	A raw vector: the contents of a compressed 'ACER ConQuest' system file created by the put command in 'ACER ConQuest'.

Value

A raw vector: the uncompressed system file, beginning with the "uncompressed" header.

See Also

conquestr::ConQuestSys()

fisherTrnsfrm *Fisher z-transform a correlation matrix*

Description

Applies Fisher's (1915) variance-stabilising transformation to the off-diagonal elements of a correlation matrix.

Usage

```
fisherTrnsfrm(myCorMat)
```

Arguments

myCorMat	A correlation matrix (unit diagonal, no missing values).

Details

Each off-diagonal correlation is mapped to

$$z_{ij} = \operatorname{artanh}(r_{ij}) = \frac{1}{2} \ln \frac{1 + r_{ij}}{1 - r_{ij}}, \quad i \neq j.$$

For bivariate-normal data with population correlation ρ, z_{ij} is approximately normal with mean $\operatorname{artanh}(\rho)$ and variance $1/(N-3)$, independent of ρ.

Since $\operatorname{artanh}(\pm 1) = \pm\infty$, off-diagonal values with $|r_{ij}| \geq 1 - 10^{-7}$ are first truncated to $\pm(1 - 10^{-7})$ (giving $|z_{ij}| \approx 8.4$) and a warning is issued. The diagonal is left at 1 and is not meaningful on the z scale.

Value

A matrix of the same dimensions with z_{ij} off the diagonal and 1 on the diagonal.

References

Fisher, R. A. (1915). Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. *Biometrika, 10*(4), 507–521. doi:10.2307/2331838

See Also

`q3ExpCorrect()`, `steigerStat()`

`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.

Generation runs as a sequence of steps. Each step is also available as its own function so it can be checked in isolation:

1. `genRespInputs()` validates and standardises the inputs.
2. `parsePerturbations()` validates `perturbR` and puts each perturbation into a standard form (one row per perturbation).
3. `perturbItem()` builds the perturbed item parameters for each perturbation.
4. `buildPerturbMap()` decides which perturbation (if any) applies to each case-item pair, based on `groups`.
5. Category probabilities are calculated and responses drawn (in C++).
6. `genMissingMCAR()` sets responses to missing completely at random.

Use `debug = TRUE` to return the output of every step, and `genResponsesProbs()` to tabulate the response probabilities used for each case, with and without perturbation.

Usage

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

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 the abilities are uni-dimensional.
mcarP	A double indicating the proportion of missing data under the MCAR assumption.
perturbR	A list of perturbations used to model misfit, or NULL (default) for none. Each element is a list describing one perturbation of one item. Elements can be given by name (recommended) or, when unnamed, in this order:

- item (integer). Which item in itemParams is affected.
- type (string). One of
  - "discrimination" - multiplies the item's discrimination (column 4 of the item matrix) by factor, and adds pivot to the delta dot.
  - "shift" - adds pivot to the delta dot, creating a uniform shift in the category characteristic curves (e.g., DIF for a group).
- factor (double). The multiplier applied to the discrimination when the type is "discrimination". For example, if the item has discrimination of 1 and factor is 1.2, responses are generated assuming a discrimination of 1.2. Ignored when the type is "shift".
- pivot (double). For "shift", the value added to the delta dot (column 2, rows 2 to k, of the item matrix), i.e., the size of the shift. For "discrimination", its meaning depends on pivotAt.
- group (character vector, optional). The group(s) in groups whose responses are perturbed, for example "bfit" or c("g1", "g2"). Omit it, or use NULL or NA, to perturb all cases. Ignored (all cases are perturbed) when groups is NULL. When elements are unnamed, every element from the fifth onwards is treated as a group.

- `pivotAt` (string, optional, by name only). Used by "discrimination":
  - "theta" (default) - `pivot` is added to the delta dot. The perturbed and unperturbed curves of a dichotomous item then cross at theta = pivot / (a * (factor - 1)), where `a` is the item's discrimination. That is, the crossing point is on the theta scale, does not depend on the item's location, and the item location moves unless `pivot` is 0 and the item is centred at theta = 0.
  - "item" - the discrimination is changed around a point `pivot` logits from the item's location (its delta dot divided by its discrimination). At that point the category probabilities of the perturbed and unperturbed item are equal, for dichotomous and partial credit items, so the item's location does not move. Use `pivot = 0` to pivot at the item location.
- `shift` (double, optional, by name only, default 0). Used by "discrimination": a value added to the delta dot after the discrimination is changed, moving the item location without moving the discrimination pivot relative to the original item. Use the "shift" type to shift an item without changing its discrimination.

An item can appear in more than one perturbation, for example to perturb it differently for different groups. If two perturbations apply to the same case and item, the later one is used, with a warning.

`debug` A Boolean. If `TRUE`, returns a list holding the responses and the intermediate data products of each step (see Value). Responses are identical to those from `debug = FALSE` for the same seed.

Value

If `debug = FALSE` (default), a matrix, `n` cases by `i` items, of scored item responses. If `debug = TRUE`, a list with elements:

- `responses`: the matrix returned when `debug = FALSE`.
- `responsesComplete`: responses before MCAR missingness was applied.
- `missing`: a logical cases by items matrix, TRUE where `genMissingMCAR()` selected the case-item pair to be missing.
- `inputs`: the standardised inputs returned by `genRespInputs()`.
- `perturbations`: a data frame, one row per perturbation, returned by `parsePerturbations()`.
- `perturbedItems`: a list of perturbed item parameter matrices, one per row of `perturbations`.
- `perturbMap`: an integer cases by items matrix returned by `buildPerturbMap()`. 0 where the case received the unperturbed item, otherwise the row of `perturbations` that was applied.
- `draws`: the uniform deviates used to draw each response (NA where the ability is NA).
- `probs`: a list, one cases by categories matrix per item, of the category probabilities used to draw each response (perturbed where applicable).
- `probsBase`: as `probs`, but under the unperturbed item parameters.

References

Adams, R. J., & Wright, B. D. (1994). When does misfit make a difference. In Objective Measurement Theory and Practice (Vol. 2). Ablex Publishing Corporation.

See Also

simplef(), genResponsesProbs(), 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)

# perturb item 2 for group "b" only, and keep the intermediate data
myGroups <- factor(sample(c("a", "b"), 100, replace = TRUE))
myPerturbations <- list(
  list(item = 2L, type = "shift", pivot = 1, group = "b")
)
myDebug <- genResponses(
  abilities = rnorm(100), groups = myGroups, itemParams = myItems,
  perturbR = myPerturbations, debug = TRUE
)
head(genResponsesProbs(myDebug, items = 2))
```

---

genResponsesProbs *genResponsesProbs*

Description

Tabulates the response probabilities used by genResponses() for each case, item, and category, alongside the probabilities under the unperturbed item. Use it to check that perturbations were applied to the intended cases, and by the intended amount.

Usage

```
genResponsesProbs(x, items = NULL)
```

Arguments

x	The list returned by genResponses(..., debug = TRUE).
items	An integer vector of the items to include. Defaults to all items.

Value

A data frame with one row per case, item, and category, and columns: case, item, group, theta, perturbation (0 if unperturbed, otherwise the row of x$perturbations applied), category, score, prob (used to draw the response), probBase (under the unperturbed item), draw (the uniform deviate), and response (before MCAR missingness).

See Also

genResponses()

Examples

```
myItem <- matrix(c(0, 0, 0, 1, 1, 0.5, 0, 1), ncol = 4, byrow = TRUE)
myDebug <- genResponses(
  abilities = rnorm(10), itemParams = list(myItem),
  perturbR = list(list(item = 1L, type = "shift", pivot = 1)), debug = TRUE
)
genResponsesProbs(myDebug)
```

`getCqChain` *getCqChain*

Description

creates a data frame representation of the estimation chain from an MCMC model. For example the Patz estimator in ACER ConQuest. The burn is discarded and only the un-skipped iterations 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 response 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))
```

---

lcs *find the largest common substring in 2 strings*

Description

find the largest common substring in 2 strings

Usage

```
lcs(str1, str2)
```

Arguments

str1	A string.
str2	A string.

---

lcs_vec *find the largest common substring in a vector of strings*

Description

find the largest common substring in a vector of strings

Usage

```
lcs_vec(input_strings)
```

Arguments

input_strings	A vector of strings.

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, ...)

## S3 method for class 'Expected'
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
```

---

pvCor *pvCor*

Description

Weighted Pearson correlations between one measure and one or more other measures, with standard errors from replicate weights and PVs. Each measure may be latent (a set of M plausible values) or manifest (a single observed column). For each PV and each weight the correlation is computed, then combined with Rubin's rules as in `pvStat()`. See the "Latent and manifest measures" section of `pvCorMatrix()` for how the two kinds are combined and for important caveats (in particular, conditioning).

With `combine = "raw"` (default) correlations are combined on the r scale, as in the PISA and TIMSS technical reports. With `combine = "fisherZ"` each PV/replicate correlation is transformed with `atanh()`, the variance components are computed on the z scale, and the estimate and confidence limits are back-transformed with `tanh()`. The reported `se` is then the delta-method SE on the r scale, `zSe * (1 - r^2)`, and the variance components are on the z scale. The z scale is preferable when correlations are near +/-1, where the r scale is bounded and its sampling distribution is skewed.

Usage

```
pvCor(
  data,
  pvs,
  vars,
  weight = NULL,
  repWeights = NULL,
  repMethod = c("BRR", "Fay", "JK1", "JK2"),
  by = NULL,
  fayK = 0.5,
  repScale = NULL,
  combine = c("raw", "fisherZ"),
  confLevel = 0.95
)
```

Arguments

data	A data frame.
pvs	Character vector of column names for the focal measure: its PV columns in draw order (latent), or a single column (manifest).
vars	The measures to correlate with pvs. Either a character vector of manifest (observed) column names, or a list whose elements are character vectors: length 1 for a manifest variable, length M for a latent measure (PV columns in draw order). List element names are used in the variable column; unnamed manifest elements are named by their column and unnamed latent elements var1, var2, ... Every latent measure (including pvs) must have the same number of PVs.
weight	Name of the full-sample weight column. If NULL, unit weights are used.
repWeights	Character vector of replicate weight column names. If NULL, only the imputation variance is estimated and the sampling variance is NA.
repMethod	One of "BRR", "Fay", "JK1", "JK2".
by	Optional name of a grouping column; correlations are computed within each level.
fayK	Fay's factor k for repMethod = "Fay".
repScale	Optional numeric override for the replicate scale factor.
combine	Either "raw" or "fisherZ".
confLevel	Confidence level for ciLower/ciUpper (normal approximation on the combining scale).

Value

A data frame with one row per variable (and group): variable, estimate, se, samplingVar, imputationVar, ciLower, ciUpper, combine, nPVs, nReps, n; with fisherZ, also zEstimate and zSe. nPVs is 1 when both measures are manifest (the estimate is then an ordinary weighted correlation and imputationVar is 0). imputationVar is B before Rubin's correction (see the "Variance components" section of pvStat()). With combine = "raw", se^2 = samplingVar + (1 + 1/nPVs) * imputationVar. With combine = "fisherZ" the variance components are on the z scale, so the same identity holds for zSe, and se = zSe * (1 - estimate^2). Missing values are deleted pairwise per variable (only cases missing on pvs, the weights, by, or that variable are dropped). This is safe because each row is a separate correlation; use pvCorMatrix() for a matrix, which requires a common set of cases.

Examples

```
set.seed(2718)
n <- 200
df <- data.frame(ses = rnorm(n), w = runif(n, 1, 3))
for (i in 1:5) df[[paste0("pv", i)]] <- 0.4 * df$ses + rnorm(n)
for (i in 1:5) df[[paste0("pvB", i)]] <- 0.5 * df[[paste0("pv", i)]] + rnorm(n)
for (r in 1:20) df[[paste0("rw", r)]] <- df$w * ifelse(seq_len(n) %% 20 == r - 1, 0, 20 / 19)
# PVs with a manifest variable
pvCor(df, paste0("pv", 1:5), "ses", weight = "w",
      repWeights = paste0("rw", 1:20), repMethod = "JK1")
```

```
# PVs with a manifest variable and another latent measure
pvCor(df, paste0("pv", 1:5), list(ses = "ses", B = paste0("pvB", 1:5)),
      weight = "w", repWeights = paste0("rw", 1:20), repMethod = "JK1")
```

---

pvCorMatrix *pvCorMatrix*

Description

Weighted Pearson correlation matrix of several measures, with standard errors from replicate weights and PVs. Each measure may be latent (a set of M plausible values, e.g., the outcome domains in IELS, PISA or TIMSS) or manifest (a single observed column, e.g., SES or age), so matrices mixing latent and observed variables can be produced.

For each PV draw m and each weight (full-sample and replicates), the full correlation matrix is computed in C++ in one pass over the data. Each element is then combined across the M draws and replicates with Rubin's rules as in `pvStat()` (see its "Variance components" section).

Usage

```
pvCorMatrix(
  data,
  pvs,
  weight = NULL,
  repWeights = NULL,
  repMethod = c("BRR", "Fay", "JK1", "JK2"),
  by = NULL,
  fayK = 0.5,
  repScale = NULL,
  combine = c("raw", "fisherZ"),
  confLevel = 0.95,
  format = c("long", "matrix")
)
```

Arguments

`data`	A data frame.
`pvs`	A named list with one element per measure. Each element is a character vector: the PV columns in draw order for a latent measure, or a single column name for a manifest variable, e.g. `list(literacy = paste0("PV", 1:5, "LIT"), numeracy = paste0("PV", 1:5, "NUM"), ses = "ESCS")`. Unnamed manifest elements are named by their column; unnamed latent elements `dim1`, `dim2`, ...
`weight`	Name of the full-sample weight column. If `NULL`, unit weights are used.
`repWeights`	Character vector of replicate weight column names. If `NULL`, only the imputation variance is estimated and the sampling variance is `NA`.
`repMethod`	One of "BRR", "Fay", "JK1", "JK2".

by	Optional name of a grouping column; correlations are computed within each level.
fayK	Fay's factor k for repMethod = "Fay".
repScale	Optional numeric override for the replicate scale factor.
combine	Either "raw" or "fisherZ".
confLevel	Confidence level for ciLower/ciUpper (normal approximation on the combining scale).
format	Either "long" (a data frame with one row per pair of measures) or "matrix" (a list of D x D matrices).

Value

With format = "long", a data frame with columns dim1, dim2, estimate, se, samplingVar, imputationVar, ciLower, ciUpper, combine, nPVs, nReps, n (and zEstimate, zSe with combine = "fisherZ"), plus a leading by column when by is used. With format = "matrix", a list with elements estimate, se, ciLower, ciUpper (D x D symmetric matrices, diagonal 1 for estimate and ciLower/ciUpper, 0 for se), n and latent (a named logical vector, TRUE for latent measures); when by is used, a named list of such lists, one per group.

Cases with a missing value on any measure, weight or by column are dropped (listwise), so all pairs use the same cases.

Latent and manifest measures

Pairing. Latent measures are paired by position: the correlation between latent A and B uses PV1 of A with PV1 of B, PV2 with PV2, and so on. This is appropriate when the PVs for all latent measures were drawn jointly from one multidimensional model (as in ConQuest), so the m-th PV of each comes from the same draw. If the latent measures were scaled in separate (e.g., unidimensional) runs, pairing by position has no special meaning, and the correlation will generally be biased towards zero. A manifest variable is the same in every draw, so its single column is reused for each of the M draws.

Consequently, for each pair of measures:

pair	estimate	imputationVar	nPVs
latent-latent	mean over M paired draws	varies	M
latent-manifest	mean over M draws (standard PISA/TIMSS method)	varies	M
manifest-manifest	ordinary weighted statistic	0	1

If no latent measure is given, the result is the ordinary weighted correlation (or covariance) matrix with replicate standard errors.

Conditioning. PVs reproduce the correlation with a background variable without bias only if that variable (or a good proxy for it) was in the population (conditioning) model used to draw them. A correlation between PVs and a manifest variable that was *not* in the conditioning model is biased towards zero. This cannot be detected from the data; check the scaling documentation of the study.

Single columns. A measure with one column is always treated as manifest. A latent measure must therefore have at least two PVs, and all latent measures must have the same number of PVs.

Validity of the matrix. Every measure is evaluated on the same cases (listwise deletion), so each per-draw matrix is a valid (positive semi-definite) correlation matrix, and so is their average (`combine = "raw"`). With `combine = "fisherZ"`, elements are back-transformed separately and the matrix is no longer guaranteed to be positive semi-definite; a warning is given if it is not. Covariance matrices from `pvCovMatrix()` are always averaged on the raw scale.

Degenerate variables. A measure that is constant within a group (e.g., a dummy variable with only one value in that group, or the by variable itself) has no defined correlation; the affected elements are `NA` and a warning names them.

Examples

```
set.seed(2718)
n <- 300
df <- data.frame(w = runif(n, 1, 3), ses = rnorm(n))
g <- 0.5 * df$ses + rnorm(n)
for (d in c("LIT", "NUM", "SR")) for (i in 1:5) {
  df[[paste0("PV", i, d)]] <- 0.7 * g + rnorm(n, 0, 0.7)
}
for (r in 1:20) df[[paste0("rw", r)]] <- df$w * ifelse(seq_len(n) %% 20 == r - 1, 0, 20 / 19)
pvList <- lapply(c(lit = "LIT", num = "NUM", sr = "SR"), function(d) paste0("PV", 1:5, d))
pvCorMatrix(df, c(pvList, ses = "ses"), weight = "w",
            repWeights = paste0("rw", 1:20), repMethod = "JK1", format = "matrix")
```

`pvCovMatrix` *pvCovMatrix*

Description

Weighted covariance matrix of several measures, with standard errors from replicate weights and PVs. Each measure may be latent (a set of M plausible values; e.g., the variances and covariances of the outcome domains in IELS, PISA or TIMSS) or manifest (a single observed column).

As in `pvCorMatrix()`, for each PV draw and each weight the full D x D covariance matrix is computed in C++; each element is then combined across draws and replicates with Rubin's rules (see the "Variance components" section of `pvStat()`). The diagonal holds the variance of each measure and matches `pvVar()` applied to that measure's columns. See the "Latent and manifest measures" section of `pvCorMatrix()` for how latent and manifest measures are paired, and for caveats about conditioning.

Covariances use the same weighted estimator as `pvVar()`: `sum(w (x - xbar)(y - ybar)) / sum(w) * n / (n - 1)`, where n is the number of cases with positive weight. Because every element is averaged on the raw scale over the same cases, the combined matrix is positive semi-definite.

Confidence limits are normal approximations. For variances near zero, or with few replicates, they can be poor (and the lower limit can be negative), because the sampling distribution of a variance is skewed.

Usage

```
pvCovMatrix(
  data,
  pvs,
  weight = NULL,
  repWeights = NULL,
  repMethod = c("BRR", "Fay", "JK1", "JK2"),
  by = NULL,
  fayK = 0.5,
  repScale = NULL,
  confLevel = 0.95,
  format = c("long", "matrix")
)
```

Arguments

data	A data frame.
pvs	A named list with one element per measure: the PV columns in draw order for a latent measure, or a single column for a manifest variable. A single measure is allowed (giving its variance).
weight	Name of the full-sample weight column. If NULL, unit weights are used.
repWeights	Character vector of replicate weight column names. If NULL, only the imputation variance is estimated and the sampling variance is NA.
repMethod	One of "BRR", "Fay", "JK1", "JK2".
by	Optional name of a grouping column; correlations are computed within each level.
fayK	Fay's factor k for repMethod = "Fay".
repScale	Optional numeric override for the replicate scale factor.
confLevel	Confidence level for ciLower/ciUpper (normal approximation on the combining scale).
format	Either "long" (a data frame with one row per element of the lower triangle, including the diagonal) or "matrix" (a list of D x D matrices).

Value

With format = "long", a data frame with columns dim1, dim2 (equal for variances), estimate, se, samplingVar, imputationVar, ciLower, ciUpper, nPVs, nReps, n, plus a leading by column when by is used. With format = "matrix", a list with elements estimate, se, ciLower, ciUpper (D x D symmetric matrices), n and latent; when by is used, a named list of such lists, one per group. imputationVar is B before Rubin's correction, so se^2 = samplingVar + (1 + 1/nPVs) * imputationVar.

Cases with a missing value on any measure, weight or by column are dropped (listwise).

Examples

```
set.seed(2718)
n <- 300
df <- data.frame(w = runif(n, 1, 3), ses = rnorm(n))
g <- 0.5 * df$ses + rnorm(n)
for (d in c("LIT", "NUM", "SR")) for (i in 1:5) {
  df[[paste0("PV", i, d)]] <- 0.7 * g + rnorm(n, 0, 0.7)
}
for (r in 1:20) df[[paste0("rw", r)]] <- df$w * ifelse(seq_len(n) %% 20 == r - 1, 0, 20 / 19)
pvList <- lapply(c(lit = "LIT", num = "NUM", sr = "SR"), function(d) paste0("PV", 1:5, d))
pvCovMatrix(df, c(pvList, ses = "ses"), weight = "w",
            repWeights = paste0("rw", 1:20), repMethod = "JK1", format = "matrix")
```

`pvDescribe` *pvDescribe*

Description

Standard descriptive statistics (mean, SD, variance, quartiles, IQR) of plausible values with replicate SEs, optionally by group.

Usage

```
pvDescribe(data, pvs, ...)
```

Arguments

`data`	A data frame.
`pvs`	Character vector of PV column names (one dimension).
`...`	Further arguments passed to `pvStat()`.

Value

A data frame stacking the results of `pvStat()` for each statistic. `imputationVar` is the between-PV variance B before Rubin's correction, so `se^2 = samplingVar + (1 + 1/nPVs) * imputationVar`. See the "Variance components" section of `pvStat()`.

pvDiff *pvDiff*

Description

Differences in a statistic between levels of a grouping variable (e.g., girls minus boys), with standard errors that account for both sampling (replicate) and imputation (PV) variance. The difference is computed within each PV and each replicate weight, then combined with Rubin's rules, so the covariance between groups induced by the sampling design is accounted for.

Usage

```
pvDiff(data, pvs, by, pairs = NULL, ...)
```

Arguments

`data`	A data frame.
`pvs`	Character vector of PV column names (one dimension).
`by`	Name of the grouping column (required).
`pairs`	Optional list of length-2 character vectors, each `c(a, b)` giving the contrast `a - b`. Defaults to all pairwise contrasts, later level minus earlier level.
`...`	Further arguments passed to `pvStat()` (e.g., `weight`, `repWeights`, `repMethod`, `stat`, `probs`).

Value

A data frame with `contrast`, `stat`, `estimate`, `se`, `samplingVar`, `imputationVar`, `t` (estimate / se). `imputationVar` is B before Rubin's correction; `se^2 = samplingVar + (1 + 1/M) * imputationVar`, where M is the number of PVs. See the "Variance components" section of `pvStat()`.

Examples

```
set.seed(2718)
n <- 200
df <- data.frame(sex = sample(c("boy", "girl"), n, TRUE), w = runif(n, 1, 3))
for (i in 1:5) df[[paste0("pv", i)]] <- rnorm(n)
for (r in 1:20) df[[paste0("rw", r)]] <- df$w * ifelse(seq_len(n) %% 20 == r - 1, 0, 20 / 19)
pvDiff(df, paste0("pv", 1:5), by = "sex", weight = "w",
       repWeights = paste0("rw", 1:20), repMethod = "JK1",
       pairs = list(c("girl", "boy")))
```

pvIqr *pvIqr*

Description

Interquartile range (weighted Q3 - Q1) of plausible values with replicate SEs. See `pvStat()`.

Usage

```
pvIqr(data, pvs, ...)
```

Arguments

`data`	A data frame.
`pvs`	Character vector of PV column names (one dimension).
`...`	Further arguments passed to `pvStat()`.

pvMean *pvMean*

Description

Mean of plausible values with replicate SEs. See `pvStat()`.

Usage

```
pvMean(data, pvs, ...)
```

Arguments

`data`	A data frame.
`pvs`	Character vector of PV column names (one dimension).
`...`	Further arguments passed to `pvStat()`.

pvMedian *pvMedian*

Description

Median of plausible values with replicate SEs. See `pvStat()`.

Usage

```
pvMedian(data, pvs, ...)
```

Arguments

`data`	A data frame.
`pvs`	Character vector of PV column names (one dimension).
`...`	Further arguments passed to `pvStat()`.

pvQuantile *pvQuantile*

Description

Quantiles of plausible values with replicate SEs. See `pvStat()`.

Usage

```
pvQuantile(data, pvs, probs = c(0.1, 0.25, 0.5, 0.75, 0.9), ...)
```

Arguments

`data`	A data frame.
`pvs`	Character vector of PV column names (one dimension).
`probs`	Numeric vector of probabilities.
`...`	Further arguments passed to `pvStat()`.

pvRegression *pvRegression*

Description

Weighted linear regression of plausible values on background variables, with standard errors from replicate weights and PVs. For each PV and each weight (full-sample plus replicates) the model is fitted by weighted least squares; coefficients and R-squared are then combined with Rubin's rules as in `pvStat()`. The heavy lifting is done in C++ (normal equations factored once per weight and reused across PVs).

Usage

```
pvRegression(
  data,
  pvs,
  formula,
  weight = NULL,
  repWeights = NULL,
  repMethod = c("BRR", "Fay", "JK1", "JK2"),
  by = NULL,
  fayK = 0.5,
  repScale = NULL,
  confLevel = 0.95
)
```

Arguments

`data`	A data frame.
`pvs`	Character vector of PV column names (one dimension).
`formula`	A one-sided formula of predictors, e.g. `~ ses + sex`. Factors, interactions and transformations are expanded by `stats::model.matrix()`. The PVs are the outcome.
`weight`	Name of the full-sample weight column. If `NULL`, unit weights are used.
`repWeights`	Character vector of replicate weight column names. If `NULL`, only the imputation variance is estimated and the sampling variance is `NA`.
`repMethod`	One of "BRR", "Fay", "JK1", "JK2".
`by`	Optional name of a grouping column; a separate model is fitted within each level.
`fayK`	Fay's factor k for `repMethod = "Fay"`.
`repScale`	Optional numeric override for the replicate scale factor.
`confLevel`	Confidence level for `ciLower`/`ciUpper` (normal approximation).

Value

A data frame with one row per term (and group), including "R-squared": `term`, `estimate`, `se`, `t`, `ciLower`, `ciUpper`, `samplingVar`, `imputationVar`, `nPVs`, `nReps`, `n`. `imputationVar` is B before Rubin's correction, so `se^2 = samplingVar + (1 + 1/nPVs) * imputationVar` (see the "Variance components" section of `pvStat()`). The R-squared row is the PV-averaged weighted R-squared; treat its SE as approximate since R-squared is bounded.

Examples

```
set.seed(2718)
n <- 200
df <- data.frame(ses = rnorm(n), sex = sample(c("boy", "girl"), n, TRUE),
                 w = runif(n, 1, 3))
for (i in 1:5) df[[paste0("pv", i)]] <- 0.4 * df$ses + rnorm(n)
for (r in 1:20) df[[paste0("rw", r)]] <- df$w * ifelse(seq_len(n) %% 20 == r - 1, 0, 20 / 19)
pvRegression(df, paste0("pv", 1:5), ~ ses + sex, weight = "w",
             repWeights = paste0("rw", 1:20), repMethod = "JK1")
```

---

pvRegression-methods *Methods for pvRegression objects*

Description

Standard model methods for objects returned by `pvRegression()`. When the model was fitted with `by`, `coef()`, `vcov()` and `confint()` return a list with one element per group, and `predict()` uses the group-specific model for each row of `newdata` (which must then contain the `by` column). Inference uses the normal approximation, as in `pvRegression()`.

Usage

```
## S3 method for class 'pvRegression'
coef(object, ...)

## S3 method for class 'pvRegression'
vcov(object, ...)

## S3 method for class 'pvRegression'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'pvRegression'
nobs(object, ...)

## S3 method for class 'pvRegression'
predict(object, newdata, se.fit = FALSE, ...)

## S3 method for class 'pvRegression'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

```
## S3 method for class 'pvRegression'
summary(object, ...)

## S3 method for class 'summary.pvRegression'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

object, x	A pvRegression object.
...	Unused.
parm	Terms to include (names or indices).
level	Confidence level.
newdata	Data frame of predictor values.
se.fit	Logical; return standard errors of the predictions, sqrt(diag(X V X')) with V the Rubin-combined covariance from vcov().
digits	Number of significant digits to print.

---

pvSd *pvSd*

Description

Standard deviation of plausible values with replicate SEs. See pvStat().

Usage

```
pvSd(data, pvs, ...)
```

Arguments

data	A data frame.
pvs	Character vector of PV column names (one dimension).
...	Further arguments passed to pvStat().

pvStat *pvStat*

Description

Core function for estimating population statistics from plausible values (PVs) with replicate-weight standard errors. The statistic is computed for each PV under the full-sample weight and under each replicate weight. Results are combined using Rubin's rules:

- estimate: mean of the M full-sample PV estimates;
- sampling variance (U): mean over PVs of the replicate variance;
- imputation variance (B): variance of the M full-sample PV estimates;
- total error variance: U + (1 + 1/M) B.

See the "Variance components" section for how the returned columns relate to `se`.

Replicate variance for one PV is `scale * sum((theta_r - theta_0)^2)`, where `scale` depends on `repMethod`:

repMethod	scale
"BRR"	1 / R
"Fay"	1 / (R (1 - k)^2), Fay's variant of BRR with factor `fayK`
"JK1"	(R - 1) / R, delete-one jackknife
"JK2"	1, paired jackknife (e.g., TIMSS, PIRLS)

Usage

```
pvStat(
  data,
  pvs,
  weight = NULL,
  repWeights = NULL,
  repMethod = c("BRR", "Fay", "JK1", "JK2"),
  stat = "mean",
  by = NULL,
  probs = 0.5,
  fayK = 0.5,
  repScale = NULL
)
```

Arguments

`data`	A data frame.
`pvs`	Character vector of PV column names (one dimension).
`weight`	Name of the full-sample weight column. If `NULL`, unit weights are used.

repWeights	Character vector of replicate weight column names. If NULL, only the imputation variance is estimated and the sampling variance is NA.
repMethod	One of "BRR", "Fay", "JK1", "JK2".
stat	One of "mean", "var", "sd", "median", "quantile", "iqr", "sumw", or a function function(x, w) returning a single number (evaluated in R, slower).
by	Optional name of a grouping column (e.g., sex). Statistics are computed within each level; an "All" row is not added.
probs	Probability for stat = "quantile".
fayK	Fay's factor k for repMethod = "Fay".
repScale	Optional numeric override for the replicate scale factor.

Value

A data frame with one row per group: group, stat, estimate, se, samplingVar, imputationVar, nPVs, nReps, n. samplingVar is U and imputationVar is B, the between-PV variance *before* Rubin's (1 + 1/M) correction, so se^2 = samplingVar + (1 + 1/nPVs) * imputationVar, not samplingVar + imputationVar. See "Variance components".

Variance components

The error variance of a PV-based estimate has two sources: sampling (which students were sampled) and measurement (each student's proficiency is uncertain, which is represented by drawing M PVs). The returned columns are:

column	symbol	definition
samplingVar	U	mean over PVs of the replicate sampling variance
imputationVar	B	variance of the M full-sample PV estimates, var(theta_m) (divisor M - 1), *before* Rubin's corre
se		sqrt(U + (1 + 1/M) * B)

Why (1 + 1/M) B rather than B. B estimates how much the statistic varies from one PV to another. The reported estimate, however, is the *average* of only M PV estimates, not the value that would be obtained with infinitely many PVs. That average has its own simulation error, with variance B / M, which must be added to the total. Hence the measurement component is B + B/M = (1 + 1/M) B (Rubin, 1987). Put another way, U + B follows from the law of total variance; the additional B/M term corrects for using a finite number of PVs. The correction matters most when M is small: it adds 20% to B with 5 PVs and 10% with 10 PVs.

Recovering se **from the columns.**

```
res <- pvMean(...)
all.equal(res$se,
          sqrt(res$samplingVar + (1 + 1 / res$nPVs) * res$imputationVar))
```

Computing sqrt(samplingVar + imputationVar) omits the B/M term and so slightly understates the standard error. The share of error variance due to measurement is (1 + 1/M) * B / se^2.

If repWeights is NULL, samplingVar and therefore se are NA; imputationVar is still returned.

Reference: Rubin, D. B. (1987). *Multiple Imputation for Nonresponse in Surveys*. Wiley.

Examples

```
set.seed(2718)
n <- 200
df <- data.frame(sex = sample(c("boy", "girl"), n, TRUE), w = runif(n, 1, 3))
for (i in 1:5) df[[paste0("pv", i)]] <- rnorm(n)
for (r in 1:20) df[[paste0("rw", r)]] <- df$w * ifelse(seq_len(n) %% 20 == r - 1, 0, 20 / 19)
pvStat(df, paste0("pv", 1:5), "w", paste0("rw", 1:20), "JK1", "mean", by = "sex")
```

---

pvVar *pvVar*

Description

Variance of plausible values with replicate SEs. See `pvStat()`.

Usage

```
pvVar(data, pvs, ...)
```

Arguments

`data`	A data frame.
`pvs`	Character vector of PV column names (one dimension).
`...`	Further arguments passed to `pvStat()`.

---

q3ExpCorrect *Correct Yen's Q3 for its expected value*

Description

Shifts the off-diagonal elements of a matrix of residual correlations (Yen's Q_3) so that their expectation under local independence is approximately zero.

Usage

```
q3ExpCorrect(myCorMat, abilityType = c("ML", "WLE", "EAP", "PV"))
```

Arguments

`myCorMat`	A correlation matrix (unit diagonal, no missing values) of item residuals, with at least 2 items.
`abilityType`	The type of ability estimate used to compute the residuals: one of `"ML"` (default), `"WLE"`, `"EAP"` or `"PV"`. The $-1/(L-1)$ correction is applied only for `"ML"` and `"WLE"`.

Details

Yen's (1984) Q_3 for items i and j is the Pearson correlation, across persons $n = 1, \ldots, N$, of the raw item residuals

$$d_{ni} = x_{ni} - E(X_{ni} \mid \hat{\theta}_n), \qquad Q_{3,ij} = r(d_i, d_j).$$

Chou and Wang (2010) instead correlate the standardised residuals $z_{ni} = d_{ni} / \sqrt{\mathrm{Var}(X_{ni} \mid \hat{\theta}_n)}$. `steigerStat()` uses whichever residuals it is given.

When abilities are maximum-likelihood (ML) or weighted-likelihood (WLE) estimates from the same responses, the residuals are constrained, and under local independence the expected value of Q_3 is approximately $-1/(L-1)$ rather than 0, where L is the number of items (Yen, 1993; Chou & Wang, 2010). For these abilities the function removes that expectation from every off-diagonal element:

$$Q^*_{3,ij} = Q_{3,ij} + \frac{1}{L-1}, \quad i \neq j.$$

The diagonal is left at 1. Values pushed outside $[-1, 1]$ are clamped to the boundary.

The $-1/(L-1)$ expectation assumes ML/WLE abilities. EAP estimates are shrunk towards the population mean, and plausible values (PVs) are random draws from each person's posterior. Either way the residuals are less tightly constrained, so the null mean of Q_3 is closer to 0, and adding $1/(L-1)$ over-corrects and inflates apparent dependence. In a simulated Rasch example ($N = 1000$, $L = 10$) the null mean was -0.111 with ML abilities but -0.075 with EAP. For `abilityType = "EAP"` or `"PV"` the correction is therefore turned off, and the input is returned unchanged after validation. No analytic expectation is applied for these cases.

Value

A matrix of the same dimensions (and dimnames). For ML/WLE it holds the corrected off-diagonal values $Q^*_{3,ij}$; for EAP/PV it is `myCorMat` unchanged.

References

Yen, W. M. (1984). Effects of local item dependence on the fit and equating performance of the three-parameter logistic model. *Applied Psychological Measurement, 8*(2), 125–145. doi:10.1177/014662168400800201

Yen, W. M. (1993). Scaling performance assessments: Strategies for managing local item dependence. *Journal of Educational Measurement, 30*(3), 187–213. doi:10.1111/j.17453984.1993.tb00423.x

Chou, Y.-T., & Wang, W.-C. (2010). Checking dimensionality in item response models with principal component analysis on standardized residuals. *Educational and Psychological Measurement, 70*(5), 717–731. doi:10.1177/0013164410379322

See Also

`fisherTrnsfrm()`, `steigerStat()`

ReadSys *ReadSys*

Description

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

Usage

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

Arguments

myFile	A 'cqReader' (an in-memory 'ACER ConQuest' system file created by the put command in 'ACER ConQuest'), as created in ConQuestSys().
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".
Dimensions	gNDim object passed in to this call to ReadSysMini from the higher-level original call to ReadSys. This value is returned in the list returned by this function call.

N	gNCases object passed in to this call to ReadSysMini from the higher-level original call to ReadSys. This value is returned in the list returned by this function call.
NPlausibles	gNPlausibles object passed in to this call to ReadSysMini from the higher-level original call to ReadSys. This value is returned in the list returned by this function call.
isDebug	Bool. Passed in to this call to ReadSysMini from the higher-level original call to ReadSys. This value is used to trigger output of debug information to standard out.

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 negative number with something - usually NA_REAL*

Description

replace a very large negative 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 matching the search term

steigerStat *Steiger test of local independence of residuals*

Description

Tests whether the matrix of item-residual correlations (Yen's Q_3) is consistent with an identity matrix, i.e. whether the standardised residuals are mutually uncorrelated (local independence).

Usage

```
steigerStat(
  myDat,
  q3Adj = TRUE,
  fisher = TRUE,
  dfAdj = FALSE,
  tpm,
  abilityType = c("ML", "WLE", "EAP", "PV")
)
```

Arguments

myDat	A data frame or matrix of standardised residuals, persons in rows and items in columns.
q3Adj	Logical; apply the Q3 expectation correction (step 2)?
fisher	Logical; apply the Fisher z-transform (step 3)?
dfAdj	Logical; adjust the degrees of freedom for sample size, number of items and targeting using the Chou and Wang (2010) adjustment (step 5)? Recommended when N is large relative to L; calibrated only for dichotomous Rasch data with $10 \leq L \leq 60$, $100 \leq N \leq 1500$ and $t \in [-1, 0]$; a warning is issued outside this range. If TRUE, supply tpm.
tpm	A double indicating the test-person match, where 0 indicates that mean item difficulty is equal to mean person ability, and -1 indicates that mean item difficulty is 1 logit below mean person ability. Defaults to 0.
abilityType	The ability estimate used to compute the residuals: "ML" (default), "WLE", "EAP" or "PV". Passed to q3ExpCorrect(); the $-1/(L-1)$ correction is applied only for "ML" and "WLE". Ignored when q3Adj = FALSE.

Details

The input is an $N \times L$ matrix of standardised residuals

$$z_{ni} = \frac{x_{ni} - E(X_{ni})}{\sqrt{\mathrm{Var}(X_{ni})}},$$

where for a dichotomous item $E(X_{ni}) = p_{ni}$ and $\mathrm{Var}(X_{ni}) = p_{ni}(1 - p_{ni})$. The steps are:

1. **Residual correlations.** Rows with any missing value are dropped (listwise deletion) and the Pearson correlation matrix $R = [Q_{3,ij}]$ is computed.
2. **Q3 correction** (q3Adj = TRUE). For ML/WLE abilities each off-diagonal element is shifted by its approximate null expectation, $Q_{3,ij} + 1/(L-1)$. The correction assumes ML/WLE abilities; for abilityType = "EAP" or "PV" it is turned off. See q3ExpCorrect().
3. **Fisher transform** (fisher = TRUE). Each off-diagonal element is transformed, $z_{ij} = \operatorname{artanh}(r_{ij})$; see fisherTrnsfrm().
4. **Test statistic.** Following Steiger (1980), under $H_0 : P = I$ the z_{ij} are approximately independent $N(0, 1/(N-3))$, so
$$\chi^2 = (N-3) \sum_{i<j} z_{ij}^2$$
is referred to a χ^2 distribution with
$$df = \frac{L(L-1)}{2}.$$
5. **Optional df adjustment** (dfAdj = TRUE). The degrees of freedom are replaced by the empirical approximation
$$df^* = \frac{L(L-1)}{2} + \frac{N}{70} - \frac{L}{4} - 12\,t - \frac{Nt}{40} + \frac{2Lt}{3},$$

where t is the test-person match (`tpm`). Chou and Wang (2010) found that, even after the $-1/(L-1)$ correction, the Type I error rate of the Steiger statistic remained inflated in some conditions (e.g., large samples and short tests). They derived this adjustment by stepwise regression on simulated data as a simple correction that works reasonably well across a variety of conditions. The coefficients are therefore an empirical calibration, not an analytic result, and are best trusted within the range of conditions they simulated. The adjustment applies only to the Steiger statistic.

The calibration data were 32 null (Rasch-generating, Rasch-analysed) conditions with 100 replications each: dichotomous items only, test lengths $L \in \{10, 20, 40, 60\}$, sample sizes $N \in \{100, 200, 500, 1500\}$, and test-person match $t \in \{0, -1\}$, where t is mean item difficulty minus mean person ability (item difficulties drawn from $U(-2, 2)$, abilities normal with variance 1). The response variable was the observed χ^2 minus its nominal df, regressed on N, L, t and their interactions. Even with the adjustment, Type I error stayed above the acceptable range in a few conditions, most notably $N = 1500$, $L = 10$, $t = -1$ (Chou & Wang, 2010, Table 4). Results outside these ranges (e.g., $t > 0$, polytomous items, or $N > 1500$) are extrapolations. The calibration also used ML-type (Winsteps) abilities with the $-1/(L-1)$ correction applied, so with EAP or PV residuals the adjustment is untested.

If `fisher = FALSE` the statistic is computed from the untransformed correlations, which is only a reasonable approximation when all correlations are small.

Value

A list with elements:

- `Cor`: the (corrected and/or transformed) residual correlation matrix;
- `steigerStat`: the χ^2 statistic;
- `Chisq_p`: the upper-tail p-value;
- `Chisq_crit`: the 95th percentile of the reference χ^2;
- `N`, `L`, `DF`: number of persons, items and degrees of freedom.

References

Steiger, J. H. (1980). Tests for comparing elements of a correlation matrix. *Psychological Bulletin, 87*(2), 245–251. doi:10.1037/00332909.87.2.245

Fisher, R. A. (1915). Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. *Biometrika, 10*(4), 507–521. doi:10.2307/2331838

Yen, W. M. (1984). Effects of local item dependence on the fit and equating performance of the three-parameter logistic model. *Applied Psychological Measurement, 8*(2), 125–145. doi:10.1177/014662168400800201

Chou, Y.-T., & Wang, W.-C. (2010). Checking dimensionality in item response models with principal component analysis on standardized residuals. *Educational and Psychological Measurement, 70*(5), 717–731. doi:10.1177/0013164410379322

See Also

`q3ExpCorrect()`, `fisherTrnsfrm()`

summariseCqChain *summariseCqChain*

Description

takes a data frame created by getCqChain and returns a list reporting the mean and variance 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

buildPerturbMap(), *13*, *15*

checkItemRespValid, 3
checkVars, 4
cnvrtItemParam, 5
coef.pvRegression
(pvRegression-methods), 51
confint.pvRegression
(pvRegression-methods), 51
ConQuestCall, 6
ConQuestRout, 7
ConQuestSys, 7
createConQuestProject, 8

DecompressSys, 9

fisherTrnsfrm, 9
fisherTrnsfrm(), *56*, *61*, *62*
fmtCqItanal, 10

genItems, 11
genMissingMCAR(), *13*, *15*
genRespInputs(), *13*, *15*
genResponses, 13
genResponses(), *13*, *16*, *68*
genResponsesProbs, 16
genResponsesProbs(), *13*, *16*
getCqChain, 17
getCqChain(), *20*
getCqData, 18
getCqDataDf, 18
getCqFit, 19
getCqHist, 20
getCqItanal, 20
getCqItanalFacility, 21
getCqItanalSummary, 22
getCqRespModel, 22
getCqTerms, 23
getCqVars, 24
ginsOnDims, 24

informationWrightMap, 25
infoWI, 26
itemInfoAtTheta, 27
itemInfoOverTheta, 27
itemListToThresholds, 28

lcs, 29
lcs_vec, 29

makeItemDfs, 30
makeItemList, 30

nobs.pvRegression
(pvRegression-methods), 51

parsePerturbations(), *13*, *15*
perturbItem(), *13*
plotCCC, 33
plotCqHist, 34
plotDif, 35
plotItemMap, 36
plotModelCCC, 37
plotModelExp, 38
plotRout, 39
predict.pvRegression
(pvRegression-methods), 51
print.pvRegression
(pvRegression-methods), 51
print.summary.pvRegression
(pvRegression-methods), 51
pvCor, 40
pvCorMatrix, 42
pvCorMatrix(), *40*, *41*, *44*
pvCovMatrix, 44
pvCovMatrix(), *44*
pvDescribe, 46
pvDiff, 47
pvIqr, 48
pvMean, 48
pvMedian, 49

pvQuantile, 49
pvRegression, 50
pvRegression(), *51*
pvRegression-methods, 51
pvSd, 52
pvStat, 53
pvStat(), *40–42*, *44*, *46–52*, *55*
pvVar, 55
pvVar(), *44*

q3ExpCorrect, 55
q3ExpCorrect(), *10*, *61*, *62*

ReadSys, 57
ReadSysMini, 57
recodeResps, 58
replaceInDataFrame, 59
replaceInVector, 59

searchConQuestSys, 60
simplef(), *13*, *16*, *68*
stats::model.matrix(), *50*
steigerStat, 60
steigerStat(), *10*, *56*
summariseCqChain, 63
summary.pvRegression
(pvRegression-methods), 51
sysFileOk, 63
sysToBMatrixDf, 64
sysToItemDifDf, 65

testInfoAtTheta, 65
testInfoOverTheta, 66
thrstThrsh, 66
transformPvs, 67

vcov(), *52*
vcov.pvRegression
(pvRegression-methods), 51

writeImportXsi, 68