Package 'Qval'

May 7, 2026

Type Package

Title The Q-Matrix Validation Methods Framework

Version 1.2.4

Date 2025-11-20

Author Haijiang Qin [aut, cre, cph] (ORCID:
<https://orcid.org/0009-0000-6721-5653>),
Lei Guo [aut, cph] (ORCID: <https://orcid.org/0000-0002-8273-3587>)

Maintainer Haijiang Qin <haijiang133@outlook.com>

Description Provide a variety of Q-matrix validation methods for the generalized cognitive diagnosis models, including the method based on the generalized deterministic input, noisy, and gate model (G-DINA) by de la Torre (2011) <DOI:10.1007/s11336-011-9207-7> discrimination index (the GDI method) by de la Torre and Chiu (2016) <DOI:10.1007/s11336-015-9467-8>, the Hull method by Najera et al. (2021) <DOI:10.1111/bmsp.12228>, the stepwise Wald test method (the Wald method) by Ma and de la Torre (2020) <DOI:10.1111/bmsp.12156>, the multiple logistic regression-based Q-matrix validation method (the MLR-B method) by Tu et al. (2022) <DOI:10.3758/s13428-022-01880-x>, the beta method based on signal detection theory by Li and Chen (2024) <DOI:10.1111/bmsp.12371> and Q-matrix validation based on relative fit index by Chen et al. (2013) <DOI:10.1111/j.1745-3984.2012.00185.x>. Different research methods and iterative procedures during Q-matrix validating are available <DOI:10.3758/s13428-024-02547-5>.

License GPL-3

Depends R (>= 4.1.0)

Imports glmnet, GDINA, MASS, Matrix, nloptr, Rcpp, parallel, plyr, gtools

LinkingTo Rcpp

RoxygenNote 7.3.3

Encoding UTF-8

NeedsCompilation yes

Collate 'att.hierarchy.R' 'CDM.R' 'IndexBeta.R' 'IndexPriority.R' 'IndexPVAF.R' 'IndexR2.R' 'is.Qident.R' 'MLR.R' 'MLRlasso.R' 'Mmatrix.R' 'QvalBeta.R' 'QvalGDI.R' 'QvalHull.R' 'QvalMLRB.R' 'QvalWald.R' 'QvalIndex.R' 'QvalValidation.R' 'RcppExports.R' 'Rmatrix.R' 'S3Extract.R' 'S3Plot.R' 'S3Print.R' 'S3Summary.R' 'S3Update.R' 'sim.Q.R' 'sim.MQ.R' 'sim.data.R' 'convex.R' 'fit.R' 'Wald.test.R' 'utils.R' 'zzz.R'

Repository CRAN

URL https://haijiangqin.com/Qval/

Date/Publication 2025-11-25 12:12:18 UTC

Contents

`att.hierarchy` *Iterative Attribute Hierarchy Exploration Methods for Cognitive Diagnosis Models*

Description

This function implements an exploratory method for attribute hierarchy structure (Zhang et al., 2025). The procedure is iterative: in each step any structural parameter that is

- less than or equal to the pre-specified threshold `eps`, or
- not greater than 0 (tested by z-statistic under Bonferroni correction and standard erros are from XPD information matrix)

is fixed to zero and the remaining parameters are re-estimated. Upon convergence, all structural parameters are both greater than `eps` and significantly larger than 0. The attribute hierarchy is then inferred from the set of parameters that remain significantly positive.

Usage

```
att.hierarchy(
  Y,
  Q,
  model = "GDINA",
  mono.constraint = FALSE,
  maxitr = 20,
  eps = 1e-07,
  alpha.level = 0.01,
  verbose = TRUE
)
```

Arguments

`Y`	A required $N \times I$ matrix or `data.frame` consisting of the responses of `N` individuals to I items. Missing values need to be coded as `NA`.
`Q`	A required binary $I \times K$ matrix containing the attributes not required or required master the items. The `i`th row of the matrix is a binary indicator vector indicating which attributes are not required (coded by 0) and which attributes are required (coded by 1) to master item i.
`model`	Type of model to be fitted; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"`, `"ACDM"`, `"LLM"`, or `"rRUM"`. Default = `"GDINA"`.
`mono.constraint`	Logical indicating whether monotonicity constraints should be fulfilled in estimation. Default = `FALSE`.
`maxitr`	Number of max iterations. Default = `20`.
`eps`	Cut-off points of the minimum vallue of structural parameters.
`alpha.level`	alpha level for the z-statistic test under Bonferroni correction. Default = `0.01`.
`verbose`	Logical indicating to print iterative information or not. Default is `TRUE`

Value

An object of class `att.hierarchy` containing the following components:

`statistic` A 4-column `matrix` for each structural parameter that is significantly larger than 0: the parameter estimate, its standard error (SE), the corresponding z-statistic, and the p-values after Bonferroni correction.

`noSig` A `logical` scalar: `TRUE` if, during iteration, all structural parameters are not greater than 0; otherwise `FALSE`.

`isNonverge` A `logical` scalar: `TRUE` if convergence was achieved within `maxitr` iterations; `FALSE` if the algorithm did not converged.

`pattern` The attribute mastery pattern matrix that contains every possible attribute mastery pattern.

`arguments` A list that stores all input arguments supplied by the user.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

Zhang, X., Jiang, Y., Xin, T., & Liu, Y. (2025). Iterative Attribute Hierarchy Exploration Methods for Cognitive Diagnosis Models. Journal of Educational and Behavioral Statistics, 50(4), 682-713. https://doi.org/10.3102/10769986241268906

`CDM` *Parameter Estimation for Cognitive Diagnosis Models (CDMs) by MMLE/EM or MMLE/BM Algorithm.*

Description

A function to estimate parameters for cognitive diagnosis models by MMLE/EM (de la Torre, 2009; de la Torre, 2011) or MMLE/BM (Ma & Jiang, 2020) algorithm. The function imports various functions from the `GDINA` package, parameter estimation for Cognitive Diagnostic Models (CDMs) was performed and extended. The `CDM` function not only accomplishes parameter estimation for most commonly used models (e.g., `GDINA`, `DINA`, `DINO`, `ACDM`, `LLM`, or `rRUM`). Furthermore, it incorporates Bayes modal estimation (BM; Ma & Jiang, 2020) to obtain more reliable estimation results, especially in small sample sizes. The monotonic constraints are able to be satisfied.

Usage

```
CDM(
  Y,
  Q,
  model = "GDINA",
  method = "EM",
  att.str = NULL,
  mono.constraint = FALSE,
```

```
  maxitr = 2000,
  verbose = 1
)
```

Arguments

Y	A required $N \times I$ matrix or data.frame consisting of the responses of N individuals to $N \times I$ items. Missing values need to be coded as NA.
Q	A required binary $I \times K$ matrix containing the attributes not required or required master the items. The ith row of the matrix is a binary indicator vector indicating which attributes are not required (coded by 0) and which attributes are required (coded by 1) to master item i.
model	Type of model to be fitted; can be "GDINA", "LCDM", "DINA", "DINO", "ACDM", "LLM", or "rRUM". Default = "GDINA".
method	Type of method to estimate CDMs' parameters; one out of "EM" and "BM". Default = "EM". However, "BM" is only available when method = "GDINA".
att.str	Specify attribute structures. NULL, by default, means there is no structure. Attribute structure needs be specified as a list - which will be internally handled by att.structure function. See examples. It can also be a matrix giving all permissible attribute profiles.
mono.constraint	Logical indicating whether monotonicity constraints should be fulfilled in estimation. Default = FALSE.
maxitr	A vector for each item or nonzero category, or a scalar which will be used for all items to specify the maximum number of EM or BM cycles allowed. Default = 2000.
verbose	Can be 0, 1 or 2, indicating to print no information, information for current iteration, or information for all iterations. Default = 1.

Details

CDMs are statistical models that fully integrates cognitive structure variables, which define the response probability of examinees on items by assuming the mechanism between attributes. In the dichotomous test, this probability is the probability of answering correctly. According to the specificity or generality of CDM assumptions, it can be divided into reduced CDM and saturated CDM.

Reduced CDMs possess specific assumptions about the mechanisms of attribute interactions, leading to clear interactions between attributes. Representative reduced models include the Deterministic Input, Noisy and Gate (DINA) model (Haertel, 1989; Junker & Sijtsma, 2001; de la Torre & Douglas, 2004), the Deterministic Input, Noisy or Gate (DINO) model (Templin & Henson, 2006), and the Additive Cognitive Diagnosis Model (A-CDM; de la Torre, 2011), the reduced Reparametrized Unified Model (rRUM; Hartz, 2002), among others. Compared to reduced models, saturated models, such as the Log-Linear Cognitive Diagnosis Model (LCDM; Henson et al., 2009) and the general Deterministic Input, Noisy and Gate model (G-DINA; de la Torre, 2011), do not have strict assumptions about the mechanisms of attribute interactions. When appropriate constraints are applied, saturated models can be transformed into various reduced models (Henson et al., 2008; de la Torre, 2011).

The LCDM is a saturated CDM fully proposed within the framework of cognitive diagnosis. Unlike reduced models that only discuss the main effects of attributes, it also considers the interaction between attributes, thus having more generalized assumptions about attributes. Its definition of the probability of correct response is as follows:

$$P(X_{pi}=1|\boldsymbol{\alpha}_l)=\frac{\exp\left[\lambda_{i0}+\boldsymbol{\lambda}_i^T\boldsymbol{h}(\boldsymbol{q}_i,\boldsymbol{\alpha}_l)\right]}{1+\exp\left[\lambda_{i0}+\boldsymbol{\lambda}_i^T\boldsymbol{h}(\boldsymbol{q}_i,\boldsymbol{\alpha}_l)\right]}$$

$$\boldsymbol{\lambda}_i^T\boldsymbol{h}(\boldsymbol{q}_i,\boldsymbol{\alpha}_l)=\sum_{k=1}^{K^*}\lambda_{ik}\alpha_{lk}+\sum_{k=1}^{K^*-1}\sum_{k'=k+1}^{K^*}\lambda_{ikk'}\alpha_{lk}\alpha_{lk'}+\cdots+\lambda_{12\cdots K^*}\prod_{k=1}^{K^*}\alpha_{lk}$$

Where, $P(X_{pi}=1|\boldsymbol{\alpha}_l)$ represents the probability of an examinee with attribute mastery pattern $\boldsymbol{\alpha}_l$ ($l=1,2,\cdots,L$ and $L=2^{K^*}$) correctly answering item i. Here, $K^*=\sum_{k=1}^{K}q_{ik}$ denotes the number of attributes in the collapsed q-vector, λ_{i0} is the intercept parameter, and $\boldsymbol{\lambda}_i=(\lambda_{i1},\lambda_{i2},\cdots,\lambda_{i12},\cdots,\lambda_{i12\cdots K^*})$ represents the effect vector of the attributes. Specifically, λ_{ik} is the main effect of attribute k, $\lambda_{ikk'}$ is the interaction effect between attributes k and k', and $\lambda_{j12\cdots K^*}$ represents the interaction effect of all required attributes.

The G-DINA, proposed by de la Torre (2011), is another saturated model that offers three types of link functions: identity link, log link, and logit link, which are defined as follows:

$$P(X_{pi}=1|\boldsymbol{\alpha}_l)=\delta_{i0}+\sum_{k=1}^{K^*}\delta_{ik}\alpha_{lk}+\sum_{k=1}^{K^*-1}\sum_{k'=k+1}^{K^*}\delta_{ikk'}\alpha_{lk}\alpha_{lk'}+\cdots+\delta_{12\cdots K^*}\prod_{k=1}^{K^*}\alpha_{lk}$$

$$log\left[P(X_{pi}=1|\boldsymbol{\alpha}_l)\right]=v_{i0}+\sum_{k=1}^{K^*}v_{ik}\alpha_{lk}+\sum_{k=1}^{K^*-1}\sum_{k'=k+1}^{K^*}v_{ikk'}\alpha_{lk}\alpha_{lk'}+\cdots+v_{12\cdots K^*}\prod_{k=1}^{K^*}\alpha_{lk}$$

$$logit\left[P(X_{pi}=1|\boldsymbol{\alpha}_l)\right]=\lambda_{i0}+\sum_{k=1}^{K^*}\lambda_{ik}\alpha_{lk}+\sum_{k=1}^{K^*-1}\sum_{k'=k+1}^{K^*}\lambda_{ikk'}\alpha_{lk}\alpha_{lk'}+\cdots+\lambda_{12\cdots K^*}\prod_{k=1}^{K^*}\alpha_{lk}$$

Where δ_{i0}, v_{i0}, and λ_{i0} are the intercept parameters for the three link functions, respectively; δ_{ik}, v_{ik}, and λ_{ik} are the main effect parameters of α_{lk} for the three link functions, respectively; $\delta_{ikk'}$, $v_{ikk'}$, and $\lambda_{ikk'}$ are the interaction effect parameters between α_{lk} and $\alpha_{lk'}$ for the three link functions, respectively; and $\delta_{i12\cdots K^*}$, $v_{i12\cdots K^*}$, and $\lambda_{i12\cdots K^*}$ are the interaction effect parameters of $\alpha_{l1}\cdots\alpha_{lK^*}$ for the three link functions, respectively. It can be observed that when the logit link is adopted, the G-DINA model is equivalent to the LCDM model.

Specifically, the A-CDM can be formulated as:

$$P(X_{pi}=1|\boldsymbol{\alpha}_l)=\delta_{i0}+\sum_{k=1}^{K^*}\delta_{ik}\alpha_{lk}$$

The rRUM, can be written as:

$$log\left[P(X_{pi}=1|\boldsymbol{\alpha}_l)\right]=\lambda_{i0}+\sum_{k=1}^{K^*}\lambda_{ik}\alpha_{lk}$$

The item response function for the linear logistic model (LLM) can be given by:

$$logit\left[P(X_{pi}=1|\boldsymbol{\alpha}_l)\right]=\lambda_{i0}+\sum_{k=1}^{K^*}\lambda_{ik}\alpha_{lk}$$

In the DINA model, every item is characterized by two key parameters: guessing (g) and slip (s). Within the traditional framework of DINA model parameterization, a latent variable η, specific to examinee p who has the attribute mastery pattern $\boldsymbol{\alpha}_l$ and responses to i, is defined as follows:

$$\eta_{li}=\prod_{k=1}^{K}\alpha_{lk}^{q_{ik}}$$

If examinee p whose attribute mastery pattern is $\boldsymbol{\alpha}_l$ has acquired every attribute required by item i, η_{pi} is given a value of 1. If not, η_{pi} is set to 0. The DINA model's item response function can be concisely formulated as such:

$$P(X_{pi}=1|\boldsymbol{\alpha}_l)=(1-s_j)^{\eta_{li}}g_j^{(1-\eta_{li})}=\delta_{i0}+\delta_{i12\cdots K}\prod_{k=1}^{K^*}\alpha_{lk}$$

$(1-s_j)^{\eta_{li}}g_j^{(1-\eta_{li})}$ is the original expression of the DINA model, while $\delta_{i0}+\delta_{i12\cdots K}\prod_{k=1}^{K^*}\alpha_{lk}$ is an equivalent form of the DINA model after adding constraints in the G-DINA model. Here, $g_j=\delta_{i0}$ and $1-s_j=\delta_{i0}+\delta_{i12\cdots K}\prod_{k=1}^{K^*}\alpha_{lk}$.

In contrast to the DINA model, the DINO model suggests that an examinee can correctly respond to an item if he/she have mastered at least one of the item's measured attributes. Additionally, like the DINA model, the DINO model also accounts for parameters related to guessing and slipping. Therefore, the main difference between DINO and DINA lies in their respective η_{li} formulations. The DINO model can be given by:

$$\eta_{li}=1-\prod_{k=1}^{K}(1-\alpha_{lk})^{q_{lk}}$$

Value

An object of class `CDM` containing the following components:

`analysis.obj` An `GDINA` object gained from `GDINA` package or an `list` after BM algorithm, depending on which estimation is used.

`alpha` Individuals' attribute parameters calculated by EAP method

`P.alpha.Xi` Individual's posterior probability

`alpha.P` Individuals' marginal mastery probabilities matrix

`P.alpha` Attribute prior weights for calculating marginalized likelihood in the last iteration

`model.fit` Some basic model-fit indices, including `Deviance`, `npar`, `AIC`, `BIC`. @seealso `fit`

`pattern` The attribute mastery pattern matrix containing all possible attribute mastery pattern.

`arguments` A list containing all input arguments

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

de la Torre, J. (2009). DINA Model and Parameter Estimation: A Didactic. Journal of Educational and Behavioral Statistics, 34(1), 115-130. DOI: 10.3102/1076998607309474.

de la Torre, J., & Douglas, J. A. (2004). Higher-order latent trait models for cognitive diagnosis. Psychometrika, 69(3), 333-353. DOI: 10.1007/BF02295640.

de la Torre, J. (2011). The Generalized DINA Model Framework. Psychometrika, 76(2), 179-199. DOI: 10.1007/s11336-011-9207-7.

Haertel, E. H. (1989). Using restricted latent class models to map the skill structure of achievement items. Journal of Educational Measurement, 26(4), 301-323. DOI: 10.1111/j.1745-3984.1989.tb00336.x.

Hartz, S. M. (2002). A Bayesian framework for the unified model for assessing cognitive abilities: Blending theory with practicality (Unpublished doctoral dissertation). University of Illinois at Urbana-Champaign.

Henson, R. A., Templin, J. L., & Willse, J. T. (2008). Defining a Family of Cognitive Diagnosis Models Using Log-Linear Models with Latent Variables. Psychometrika, 74(2), 191-210. DOI: 10.1007/s11336-008-9089-5.

Huebner, A., & Wang, C. (2011). A note on comparing examinee classification methods for cognitive diagnosis models. Educational and Psychological Measurement, 71, 407-419. DOI: 10.1177/0013164410388832.

Junker, B. W., & Sijtsma, K. (2001). Cognitive assessment models with few assumptions, and connections with nonparametric item response theory. Applied Psychological Measurement, 25(3), 258-272. DOI: 10.1177/01466210122032064.

Ma, W., & Jiang, Z. (2020). Estimating Cognitive Diagnosis Models in Small Samples: Bayes Modal Estimation and Monotonic Constraints. Applied Psychological Measurement, 45(2), 95-111. DOI: 10.1177/0146621620977681.

Templin, J. L., & Henson, R. A. (2006). Measurement of psychological disorders using cognitive diagnosis models. Psychological methods, 11(3), 287-305. DOI: 10.1037/1082-989X.11.3.287.

Tu, D., Chiu, J., Ma, W., Wang, D., Cai, Y., & Ouyang, X. (2022). A multiple logistic regression-based (MLR-B) Q-matrix validation method for cognitive diagnosis models: A confirmatory approach. Behavior Research Methods. DOI: 10.3758/s13428-022-01880-x.

See Also

validation.

Examples

```
#####################################################################
#                           Example 1                               #
#            fit using MMLE/EM to fit the GDINA models              #
#####################################################################
set.seed(123)
```

```
library(Qval)

## generate Q-matrix and data to fit
K <- 3
I <- 30
Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
data.obj <- sim.data(Q = Q, N = 500, IQ = IQ,
                         model = "GDINA", distribute = "mvnorm")


## using MMLE/EM to fit GDINA model
CDM.obj <- CDM(data.obj$dat, Q, model = "GDINA",
                       method = "EM", maxitr = 2000, verbose = 1)



#####################################################################
#                           Example 2                               #
#               fit using MMLE/BM to fit the DINA                   #
#####################################################################

set.seed(123)

library(Qval)

## generate Q-matrix and data to fit
K <- 5
I <- 30
Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
data.obj <- sim.data(Q = Q, N = 500, IQ = IQ,
                 model = "DINA", distribute = "horder")

## using MMLE/BM to fit GDINA model
CDM.obj <- CDM(data.obj$dat, Q, model = "GDINA",
               method = "BM", maxitr = 1000, verbose = 2)


#####################################################################
#                           Example 3                               #
# fit using MMLE/EM to fit the GDINA with attribute structures      #
#####################################################################

set.seed(123)

library(Qval)
```

```
## generate Q-matrix and data to fit
K <- 3
I <- 30
att.str <- list(c(1,2),
                c(2,3))
Q <- sim.Q(K, I, att.str)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
data.obj <- sim.data(Q = Q, N = 500, IQ = IQ, att.str=att.str,
                     model = "GDINA")

## using MMLE/EM to fit GDINA model with attribute structures
CDM.obj <- CDM(data.obj$dat, Q, model = "GDINA", att.str=att.str,
                method = "EM", maxitr = 2000, verbose = 1)

summary(CDM.obj)
```

`extract` *Extract Components from Qval Package Objects*

Description

A unified extractor function for retrieving internal components from objects produced by the **Qval** package. This method allows users to access key elements such as model results, validation logs, and simulation settings in a structured and object-oriented manner.

Usage

```
extract(object, what, ...)

## S3 method for class 'CDM'
extract(object, what, ...)

## S3 method for class 'validation'
extract(object, what, ...)

## S3 method for class 'sim.data'
extract(object, what, ...)

## S3 method for class 'fit'
extract(object, what, ...)

## S3 method for class 'is.Qident'
extract(object, what, ...)
```

```
## S3 method for class 'att.hierarchy'
extract(object, what, ...)
```

Arguments

object	An object of class CDM, validation, sim.data, fit, is.Qident, att.hierarchy.
what	A character string specifying the name of the component to extract.
...	Additional arguments (currently ignored).

Details

This generic extractor supports three core object classes: CDM, validation, sim.data, fit, is.Qident, att.hierarchy. It is intended to streamline access to commonly used internal components without manually referencing object slots. The available components for each class are listed below:

CDM Cognitive Diagnosis Model fitting results. Available components:

- analysis.obj The internal model fitting object (e.g., GDINA or Baseline Model).
- alpha Estimated attribute profiles (EAP estimates) for each respondent.
- P.alpha.Xi Posterior distribution of latent attribute patterns.
- alpha.P Marginal attribute mastery probabilities (estimated).
- P.alpha Prior attribute probabilities at convergence.
- pattern The attribute mastery pattern matrix containing all possible attribute mastery pattern.
- Deviance Negative twice the marginal log-likelihood (model deviance).
- npar Number of free parameters estimated in the model.
- AIC Akaike Information Criterion.
- BIC Bayesian Information Criterion.
- call The original model-fitting function call.
- ... Can extract corresponding value from the GDINA object.

validation Q-matrix validation results. Available components:

- Q.orig The original Q-matrix submitted for validation.
- Q.sug The suggested (revised) Q-matrix after validation.
- time.cost Total computation time for the validation procedure.
- process Log of Q-matrix modifications across iterations.
- iter Number of iterations performed during validation.
- priority Attribute priority matrix (available for PAA-based methods only).
- Hull.fit Data required to plot the Hull method results (for Hull-based validation only).
- call The original function call used for validation.

sim.data Simulated data and parameters used in cognitive diagnosis simulation studies:

- dat Simulated dichotomous response matrix ($N \times I$).
- Q Q-matrix used to generate the data.
- attribute True latent attribute profiles ($N \times K$).
- catprob.parm Item-category conditional success probabilities (list format).

`delta.parm` Item-level delta parameters (list format).

`higher.order.parm` Higher-order model parameters (if used).

`mvnorm.parm` Parameters for the multivariate normal attribute distribution (if used).

`LCprob.parm` Latent class-based success probability matrix.

`call` The original function call that generated the simulated data.

`fit` Relative fit indices (-2LL, AIC, BIC, CAIC, SABIC) and absolute fit indices (M_2 test, $RMSEA_2$, SRMSR):

`npar` The number of parameters.

`-2LL` The Deviance.

`AIC` The Akaike information criterion.

`BIC` The Bayesian information criterion.

`CAIC` The consistent Akaike information criterion.

`SABIC` The Sample-size Adjusted BIC.

`M2` A vector consisting of M_2 statistic, degrees of freedom, significance level, and $RMSEA_2$ (Liu, Tian, & Xin, 2016).

`SRMSR` The standardized root mean squared residual (SRMSR; Ravand & Robitzsch, 2018).

`is.Qident` Results of whether the Q-matrix is identifiable:

`completeness` TRUE if $K \times K$ identity submatrix exists.

`distinctness` TRUE if remaining columns are distinct.

`repetition` TRUE if every attribute appears more than 3 items.

`genericCompleteness` TRUE if two different generic complete $K \times K$ submatrices exist.

`genericRepetition` TRUE if at least one '1' exists outside those submatrices.

`Q1, Q2` Identified generic complete submatrices (if found).

`Q.star` Remaining part after removing rows in `Q1` and `Q2`.

`locallyGenericIdentifiability` TRUE if local generic identifiability holds.

`globallyGenericIdentifiability` TRUE if global generic identifiability holds.

`Q.reconstructed.DINA` Reconstructed Q-matrix with low-frequency attribute moved to first column.

`att.hierarchy` Results of iterative attribute hierarchy exploration:

`noSig` TRUE all structural parameters are not greater than 0.

`isNonverge` TRUE if convergence was achieved.

`statistic` A 4-column `data.frame` results for each structural parameter that is significantly larger than 0.

`pattern` The attribute pattern matrix under iterative attribute hierarchy.

Value

The requested component. The return type depends on the specified `what` and the class of the `object`.

Methods (by class)

- `extract(CDM)`: Extract fields from a `CDM` object
- `extract(validation)`: Extract fields from a `validation` object
- `extract(sim.data)`: Extract fields from a `sim.data` object
- `extract(fit)`: Extract fields from a `fit` object
- `extract(is.Qident)`: Extract fields from a `is.Qident` object
- `extract(att.hierarchy)`: Extract fields from a `att.hierarchy` object

References

Khaldi, R., Chiheb, R., & Afa, A.E. (2018). Feed-forward and Recurrent Neural Networks for Time Series Forecasting: Comparative Study. In: Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications (LOPAL 18). Association for Computing Machinery, New York, NY, USA, Article 18, 1–6. DOI: 10.1145/3230905.3230946.

Liu, Y., Tian, W., & Xin, T. (2016). An application of M2 statistic to evaluate the fit of cognitive diagnostic models. Journal of Educational and Behavioral Statistics, 41, 3–26. DOI: 10.3102/1076998615621293.

Ravand, H., & Robitzsch, A. (2018). Cognitive diagnostic model of best choice: a study of reading comprehension. Educational Psychology, 38, 1255–1277. DOI: 10.1080/01443410.2018.1489524.

Examples

```
library(Qval)
set.seed(123)


#####################################################################
# Example 1: sim.data extraction                                    #
#####################################################################
Q <- sim.Q(3, 10)
data.obj <- sim.data(Q, N = 200)
extract(data.obj, "dat")


#####################################################################
# Example 2: CDM extraction                                         #
#####################################################################
CDM.obj <- CDM(data.obj$dat, Q)
extract(CDM.obj, "alpha")
extract(CDM.obj, "AIC")


#####################################################################
# Example 3: validation extraction                                  #
#####################################################################
validation.obj <- validation(data.obj$dat, Q, CDM.obj)
Q.sug <- extract(validation.obj, "Q.sug")
print(Q.sug)
```

```
#################################################################
# Example 4: fit extraction                                     #
#################################################################
fit.obj <- fit(data.obj$dat, Q.sug, model="GDINA")
extract(fit.obj, "M2")
```

`fit` *Calculate Fit Indices*

Description

Calculate relative fit indices (-2LL, AIC, BIC, CAIC, SABIC) and absolute fit indices (M_2 test, $RMSEA_2$, SRMSR) using the `modelfit` function in the `GDINA` package.

Usage

```
fit(
  Y,
  Q,
  model = "GDINA",
  att.str = NULL,
  mono.constraint = FALSE,
  maxitr = 2000,
  verbose = 1
)
```

Arguments

`Y`	A required $N \times I$ matrix or `data.frame` consisting of the responses of `N` individuals to I items. Missing values need to be coded as `NA`.
`Q`	A required binary $I \times K$ matrix containing the attributes not required or required master the items. The `i`th row of the matrix is a binary indicator vector indicating which attributes are not required (coded by 0) and which attributes are required (coded by 1) to master item i.
`model`	Type of model to be fitted; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"`, `"ACDM"`, `"LLM"`, or `"rRUM"`. Default = `"GDINA"`.
`att.str`	Specify attribute structures. `NULL`, by default, means there is no structure. Attribute structure needs be specified as a list - which will be internally handled by `att.structure` function. See examples. It can also be a matrix giving all permissible attribute profiles.
`mono.constraint`	Logical indicating whether monotonicity constraints should be fulfilled in estimation. Default = `FALSE`.
`maxitr`	A vector for each item or nonzero category, or a scalar which will be used for all items to specify the maximum number of EM cycles allowed. Default = `2000`.

verbose Can be 0, 1 or 2, indicating to print no information, information for current iteration, or information for all iterations. Default = 1.

Value

An object of class fit. The fit contains various fit indices:

npar The number of parameters.

-2LL The Deviance.

AIC The Akaike information criterion.

BIC The Bayesian information criterion.

CAIC The consistent Akaike information criterion.

SABIC The Sample-size Adjusted BIC.

M2 A vector consisting of M_2 statistic, degrees of freedom, significance level, and $RMSEA_2$ (Liu, Tian, & Xin, 2016).

SRMSR The standardized root mean squared residual (SRMSR; Ravand & Robitzsch, 2018).

arguments A list containing all input arguments

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

Khaldi, R., Chiheb, R., & Afa, A.E. (2018). Feed-forward and Recurrent Neural Networks for Time Series Forecasting: Comparative Study. In: Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications (LOPAL 18). Association for Computing Machinery, New York, NY, USA, Article 18, 1–6. DOI: 10.1145/3230905.3230946.

Liu, Y., Tian, W., & Xin, T. (2016). An application of M2 statistic to evaluate the fit of cognitive diagnostic models. Journal of Educational and Behavioral Statistics, 41, 3–26. DOI: 10.3102/1076998615621293.

Ravand, H., & Robitzsch, A. (2018). Cognitive diagnostic model of best choice: a study of reading comprehension. Educational Psychology, 38, 1255–1277. DOI: 10.1080/01443410.2018.1489524.

Examples

```
set.seed(123)

library(Qval)

## generate Q-matrix and data to fit
K <- 5
I <- 30
Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
data <- sim.data(Q = Q, N = 500, IQ = IQ, model = "GDINA", distribute = "horder")
```

```
## calculate fit indices
fit.indices <- fit(Y = data$dat, Q = Q, model = "GDINA")
print(fit.indices)
```

`get.beta` *Calculate* β

Description

The function is able to calculate the β index for all items after fitting `CDM` or directly.

Usage

```
get.beta(
  Y = NULL,
  Q = NULL,
  att.str = NULL,
  CDM.obj = NULL,
  mono.constraint = FALSE,
  model = "GDINA"
)
```

Arguments

`Y`	A required $N \times I$ matrix or `data.frame` consisting of the responses of `N` individuals to $N \times I$ items. Missing values need to be coded as `NA`.
`Q`	A required binary $I \times K$ matrix containing the attributes not required or required master the items. The `i`th row of the matrix is a binary indicator vector indicating which attributes are not required (coded by 0) and which attributes are required (coded by 1) to master item i.
`att.str`	Specify attribute structures. `NULL`, by default, means there is no structure. Attribute structure needs be specified as a list - which will be internally handled by `att.structure` function. See examples. It can also be a matrix giving all permissible attribute profiles.
`CDM.obj`	An object of class `CDM.obj`. Can be `NULL`, but when it is not `NULL`, it enables rapid validation of the Q-matrix without the need for parameter estimation. @seealso `CDM`.
`mono.constraint`	Logical indicating whether monotonicity constraints should be fulfilled in estimation. Default = `FALSE`.
`model`	Type of model to be fitted; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"`, `"ACDM"`, `"LLM"`, or `"rRUM"`. Default = `"GDINA"`.

Details

For item i with the q-vector of the c-th ($c = 1, 2, ..., 2^K$) type, the β index is computed as follows:

$$\beta_{ic} = \sum_{l=1}^{2^K} \left| \frac{r_{li}}{n_l} P_{ic}(\boldsymbol{\alpha}_l) - \left(1 - \frac{r_{li}}{n_l}\right) [1 - P_{ic}(\boldsymbol{\alpha}_l)] \right| = \sum_{l=1}^{2^K} \left| \frac{r_{li}}{n_l} - [1 - P_{ic}(\boldsymbol{\alpha}_l)] \right|$$

In the formula, r_{li} represents the number of examinees in attribute mastery pattern $\boldsymbol{\alpha}_l$ who correctly answered item i, while n_l is the total number of examinees in attribute mastery pattern $\boldsymbol{\alpha}_l$. $P_{ic}(\boldsymbol{\alpha}_l)$ denotes the probability that an examinee in attribute mastery pattern $\boldsymbol{\alpha}_l$ answers item i correctly when the q-vector for item i is of the c-th type. In fact, $\frac{r_{li}}{n_l}$ is the observed probability that an examinee in attribute mastery pattern $\boldsymbol{\alpha}_l$ answers item i correctly, and β_{jc} represents the difference between the actual proportion of correct answers for item i in each attribute mastery pattern and the expected probability of answering the item incorrectly in that state. Therefore, to some extent, β_{jc} can be considered as a measure of discriminability.

Value

An object of class `matrix`, which consisted of β index for each item and each possible attribute mastery pattern.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

Li, J., & Chen, P. (2024). A new Q-matrix validation method based on signal detection theory. British Journal of Mathematical and Statistical Psychology, 00, 1–33. DOI: 10.1111/bmsp.12371

See Also

`validation`

Examples

```
library(Qval)

set.seed(123)

## generate Q-matrix and data
K <- 4
I <- 20
Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.3),
  P1 = runif(I, 0.7, 0.9)
)

model <- "GDINA"
data <- sim.data(Q = Q, N = 500, IQ = IQ, model = model, distribute = "horder")
```

```
## calculate beta directly
beta <-get.beta(Y = data$dat, Q = Q, model = model)
print(beta)

## calculate beta after fitting CDM
CDM.obj <- CDM(data$dat, Q, model=model)
beta <-get.beta(CDM.obj = CDM.obj)
print(beta)
```

`get.Mmatrix` *Calculate* $\mathbf{M}$ *matrix*

Description

Calculate $\mathbf{M}$ matrix for saturated CDMs (de la Torre, 2011). The $\mathbf{M}$ matrix is a matrix used to represent the interaction mechanisms between attributes.

Usage

```
get.Mmatrix(K = NULL, pattern = NULL)
```

Arguments

`K`	The number of attributes. Can be `NULL` if the argument `pattern` is not `NULL`.
`pattern`	The attribute mastery pattern matrix containing all possible attribute mastery pattern. Can be gained from `attributepattern`. Also can be `NULL` if `K` is not `NULL`.

Value

An object of class `matrix`.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

de la Torre, J. (2011). The Generalized DINA Model Framework. Psychometrika, 76(2), 179-199. DOI: 10.1007/s11336-011-9207-7.

Examples

```
library(Qval)

Mmatrix <-  get.Mmatrix(K = 3)

print(Mmatrix)
```

`get.priority` *Priority of Attribute*

Description

This function will provide the priorities of attributes for all items.

Usage

```
get.priority(
  Y = NULL,
  Q = NULL,
  att.str = NULL,
  CDM.obj = NULL,
  mono.constraint = FALSE,
  model = "GDINA"
)
```

Arguments

`Y`	A required $N \times I$ matrix or `data.frame` consisting of the responses of `N` individuals to $N \times I$ items. Missing values need to be coded as `NA`.
`Q`	A required binary $I \times K$ matrix containing the attributes not required or required master the items. The `i`th row of the matrix is a binary indicator vector indicating which attributes are not required (coded by 0) and which attributes are required (coded by 1) to master item i.
`att.str`	Specify attribute structures. `NULL`, by default, means there is no structure. Attribute structure needs be specified as a list - which will be internally handled by `att.structure` function. See examples. It can also be a matrix giving all permissible attribute profiles.
`CDM.obj`	An object of class `CDM.obj`. When it is not NULL, it enables rapid validation of the Q-matrix without the need for parameter estimation. @seealso `CDM`.
`mono.constraint`	Logical indicating whether monotonicity constraints should be fulfilled in estimation. Default = `FALSE`.
`model`	Type of model to fit; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"` , `"ACDM"`, `"LLM"`, or `"rRUM"`. Default = `"GDINA"`. @seealso `CDM`.

Details

The calculation of priorities is straightforward (Qin & Guo, 2025): the priority of an attribute is the regression coefficient obtained from a LASSO multinomial logistic regression, with the attribute as the independent variable and the response data from the examinees as the dependent variable. The formula (Tu et al., 2022) is as follows:

$$\log[\frac{P(X_{pi}=1|\mathbf{\Lambda}_p)}{P(X_{pi}=0|\mathbf{\Lambda}_p)}] = logit[P(X_{pi}=1|\mathbf{\Lambda}_p)] = \beta_{i0} + \beta_{i1}\Lambda_{p1} + \ldots + \beta_{ik}\Lambda_{pk} + \ldots + \beta_{iK}\Lambda_{pK}$$

Where X_{pi} represents the response of examinee p on item i, $\mathbf{\Lambda}_p$ denotes the marginal mastery probabilities of examinee p (which can be obtained from the return value `alpha.P` of the `CDM` function), β_{i0} is the intercept term, and β_{ik} represents the regression coefficient.

The LASSO loss function can be expressed as:

$$l_{lasso}(\boldsymbol{X}_i|\mathbf{\Lambda}) = l(\boldsymbol{X}_i|\mathbf{\Lambda}) - \lambda|\boldsymbol{\beta}_i|$$

Where $l_{lasso}(\boldsymbol{X}_i|\mathbf{\Lambda})$ is the penalized likelihood, $l(\boldsymbol{X}_i|\mathbf{\Lambda})$ is the original likelihood, and λ is the tuning parameter for penalization (a larger value imposes a stronger penalty on $\boldsymbol{\beta}_i = [\beta_{i1}, \ldots, \beta_{ik}, \ldots, \beta_{iK}]$). The priority for attribute i is defined as: $\boldsymbol{priority}_i = \boldsymbol{\beta}_i = [\beta_{i1}, \ldots, \beta_{ik}, \ldots, \beta_{iK}]$

Value

A matrix containing all attribute priorities.

References

Qin, H., & Guo, L. (2025). Priority attribute algorithm for Q-matrix validation: A didactic. Behavior Research Methods, 57(1), 31. DOI: 10.3758/s13428-024-02547-5.

Tu, D., Chiu, J., Ma, W., Wang, D., Cai, Y., & Ouyang, X. (2022). A multiple logistic regression-based (MLR-B) Q-matrix validation method for cognitive diagnosis models: A confirmatory approach. Behavior Research Methods. DOI: 10.3758/s13428-022-01880-x.

See Also

`validation`

Examples

```
set.seed(123)
library(Qval)

## generate Q-matrix and data
K <- 5
I <- 20
IQ <- list(
  P0 = runif(I, 0.1, 0.3),
  P1 = runif(I, 0.7, 0.9)
)
```

```
Q <- sim.Q(K, I)
data <- sim.data(Q = Q, N = 500, IQ = IQ, model = "GDINA", distribute = "horder")
MQ <- sim.MQ(Q, 0.1)

CDM.obj <- CDM(data$dat, MQ)

priority <- get.priority(data$dat, Q, CDM.obj=CDM.obj)
head(priority)
```

`get.PVAF` *Calculate PVAF*

Description

The function is able to calculate the proportion of variance accounted for ($PVAF$) for all items after fitting `CDM` or directly.

Usage

```
get.PVAF(
  Y = NULL,
  Q = NULL,
  att.str = NULL,
  CDM.obj = NULL,
  mono.constraint = FALSE,
  model = "GDINA"
)
```

Arguments

`Y`	A required $N \times I$ matrix or `data.frame` consisting of the responses of `N` individuals to $N \times I$ items. Missing values need to be coded as `NA`.
`Q`	A required binary $I \times K$ matrix containing the attributes not required or required master the items. The `i`th row of the matrix is a binary indicator vector indicating which attributes are not required (coded by 0) and which attributes are required (coded by 1) to master item i.
`att.str`	Specify attribute structures. `NULL`, by default, means there is no structure. Attribute structure needs be specified as a list - which will be internally handled by `att.structure` function. See examples. It can also be a matrix giving all permissible attribute profiles.
`CDM.obj`	An object of class `CDM.obj`. Can be `NULL`, but when it is not `NULL`, it enables rapid verification of the Q-matrix without the need for parameter estimation.

`mono.constraint`
: Logical indicating whether monotonicity constraints should be fulfilled in estimation. Default = `FALSE`.

`model`
: Type of model to be fitted; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"`, `"ACDM"`, `"LLM"`, or `"rRUM"`. Default = `"GDINA"`.

Details

The intrinsic essence of the GDI index (as denoted by ζ_2) is the weighted variance of all 2^{K*} attribute mastery patterns' probabilities of correctly responding to item i, which can be computed as:

$$\zeta^2 = \sum_{l=1}^{2^K} \pi_l (P(X_{pi} = 1|\boldsymbol{\alpha}_l) - \bar{P}_i)^2$$

where π_l represents the prior probability of mastery pattern l; $\bar{P}_i = \sum_{k=1}^{2^K} \pi_l P(X_{pi} = 1|\boldsymbol{\alpha}_l)$ is the weighted average of the correct response probabilities across all attribute mastery patterns. When the q-vector is correctly specified, the calculated ζ^2 should be maximized, indicating the maximum discrimination of the item.

Theoretically, ζ^2 is larger when $\boldsymbol{q}_i$ is either specified correctly or over-specified, unlike when $\boldsymbol{q}_i$ is under-specified, and that when $\boldsymbol{q}_i$ is over-specified, ζ^2 is larger than but close to the value of $\boldsymbol{q}_i$ when specified correctly. The value of ζ^2 continues to increase slightly as the number of over-specified attributes increases, until $\boldsymbol{q}_i$ becomes $\boldsymbol{q}_{i1:K}$ ($\boldsymbol{q}_{i1:K}$ = [11...1]). Thus, ζ^2/ζ^2_{max} is computed to indicate the proportion of variance accounted for by $\boldsymbol{q}_i$, called the $PVAF$.

Value

An object of class `matrix`, which consisted of $PVAF$ for each item and each possible q-vector.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

de la Torre, J., & Chiu, C. Y. (2016). A General Method of Empirical Q-matrix Validation. Psychometrika, 81(2), 253-273. DOI: 10.1007/s11336-015-9467-8.

See Also

`validation`

Examples

```
library(Qval)

set.seed(123)

## generate Q-matrix and data
K <- 3
I <- 20
```

```
Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
data <- sim.data(Q = Q, N = 500, IQ = IQ, model = "GDINA", distribute = "horder")

## calculate PVAF directly
PVAF <-get.PVAF(Y = data$dat, Q = Q)
print(PVAF)

## calculate PVAF after fitting CDM
CDM.obj <- CDM(data$dat, Q, model="GDINA")
PVAF <-get.PVAF(CDM.obj = CDM.obj)
print(PVAF)
```

`get.R2` *Calculate McFadden Pseudo-R2*

Description

The function is able to calculate the McFadden pseudo-R^2 (R^2) for all items after fitting `CDM` or directly.

Usage

```
get.R2(
  Y = NULL,
  Q = NULL,
  att.str = NULL,
  CDM.obj = NULL,
  mono.constraint = FALSE,
  model = "GDINA"
)
```

Arguments

`Y`	A required $N \times I$ matrix or `data.frame` consisting of the responses of `N` individuals to $N \times I$ items. Missing values need to be coded as `NA`.
`Q`	A required binary $I \times K$ matrix containing the attributes not required or required master the items. The `i`th row of the matrix is a binary indicator vector indicating which attributes are not required (coded by 0) and which attributes are required (coded by 1) to master item i.
`att.str`	Specify attribute structures. `NULL`, by default, means there is no structure. Attribute structure needs be specified as a list - which will be internally handled by `att.structure` function. See examples. It can also be a matrix giving all permissible attribute profiles.

`CDM.obj` An object of class `CDM.obj`. Can be NULL, but when it is not NULL, it enables rapid verification of the Q-matrix without the need for parameter estimation. @seealso `CDM`.

`mono.constraint` Logical indicating whether monotonicity constraints should be fulfilled in estimation. Default = `FALSE`.

`model` Type of model to fit; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"`, `"ACDM"`, `"LLM"`, or `"rRUM"`. Default = `"GDINA"`.

Details

The McFadden pseudo-R^2 (McFadden, 1974) serves as a definitive model-fit index, quantifying the proportion of variance explained by the observed responses. Comparable to the squared multiple-correlation coefficient in linear statistical models, this coefficient of determination finds its application in logistic regression models. Specifically, in the context of the CDM, where probabilities of accurate item responses are predicted for each examinee, the McFadden pseudo-R^2 provides a metric to assess the alignment between these predictions and the actual responses observed. Its computation is straightforward, following the formula:

$$R_i^2 = 1 - \frac{\log(L_{im})}{\log(L_{i0})}$$

where $\log(L_{im})$ is the log-likelihood of the model, and $\log(L_{i0})$ is the log-likelihood of the null model. If there were N examinees taking a test comprising I items, then $\log(L_{im})$ would be computed as:

$$\log(L_{im}) = \sum_{p}^{N} \log \sum_{l=1}^{2^{K^*}} \pi(\boldsymbol{\alpha}_l^* | \boldsymbol{X}_p) P_i(\boldsymbol{\alpha}_l^*)^{X_{pi}} \left[1 - P_i(\boldsymbol{\alpha}_l^*)\right]^{(1-X_{pi})}$$

where $\pi(\boldsymbol{\alpha}_l^* | \boldsymbol{X}_p)$ is the posterior probability of examinee p with attribute mastery pattern $\boldsymbol{\alpha}_l^*$ when their response vector is $\boldsymbol{X}_p$, and X_{pi} is examinee p's response to item i. Let $\bar{X}_i$ be the average probability of correctly responding to item i across all N examinees; then $\log(L_{i0})$ could be computed as:

$$\log(L_{i0}) = \sum_{p}^{N} \log \bar{X}_i^{X_{pi}} (1 - \bar{X}_i)^{(1-X_{pi})}$$

Value

An object of class `matrix`, which consisted of R^2 for each item and each possible attribute mastery pattern.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

McFadden, D. (1974). Conditional logit analysis of qualitative choice behavior. In P. Zarembka (Ed.), Frontiers in economics (pp.105–142). Academic Press.

Najera, P., Sorrel, M. A., de la Torre, J., & Abad, F. J. (2021). Balancing ft and parsimony to improve Q-matrix validation. British Journal of Mathematical and Statistical Psychology, 74, 110–130. DOI: 10.1111/bmsp.12228.

Qin, H., & Guo, L. (2023). Using machine learning to improve Q-matrix validation. Behavior Research Methods. DOI: 10.3758/s13428-023-02126-0.

See Also

validation

Examples

```
library(Qval)

set.seed(123)

## generate Q-matrix and data
K <- 3
I <- 20
Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
data <- sim.data(Q = Q, N = 500, IQ = IQ, model = "GDINA", distribute = "horder")

## calculate R2 directly
R2 <-get.R2(Y = data$dat, Q = Q)
print(R2)

## calculate R2 after fitting CDM
CDM.obj <- CDM(data$dat, Q, model="GDINA")
R2 <-get.R2(CDM.obj = CDM.obj)
print(R2)
```

---

get.Rmatrix *Restriction Matrix*

Description

This function returns the restriction matrix (de la Torre, 2011; Ma & de la Torre, 2020) based on two q-vectors, where the two q-vectors can only differ by one attribute.

Usage

```
get.Rmatrix(q1, q2, pattern.reduced.q1 = NULL, pattern.reduced.q2 = NULL)
```

Arguments

q1	A q-vector
q2	Another q-vector
pattern.reduced.q1	reduced attribute patterns for q1 (Default = NULL)
pattern.reduced.q2	reduced attribute patterns for q2 (Default = NULL)

Value

A restriction matrix

References

de la Torre, J. (2011). The Generalized DINA Model Framework. Psychometrika, 76(2), 179-199. DOI: 10.1007/s11336-011-9207-7.

Ma, W., & de la Torre, J. (2020). An empirical Q-matrix validation method for the sequential generalized DINA model. British Journal of Mathematical and Statistical Psychology, 73(1), 142-163. DOI: 10.1111/bmsp.12156.

See Also

Wald.test

Examples

```
library(Qval)
q1 <- c(1, 1, 0)
q2 <- c(1, 1, 1)

Rmatrix <- get.Rmatrix(q1, q2)

print(Rmatrix)
```

---

is.Qident *Check Whether a Q-Matrix is Identifiable*

Description

Checks whether a given Q-matrix satisfies the conditions for **strict identifiability** and **generic identifiability** under cognitive diagnosis models (CDMs), including the DINA, DINO, and saturated models, based on theoretical results from Gu & Xu (2021).

This function evaluates both joint strict identifiability and various forms of generic identifiability, including global and local cases, by verifying structural conditions on the Q-matrix.

Usage

```
is.Qident(Q, verbose = TRUE)
```

Arguments

Q	An $I \times K$ binary Q-matrix (`matrix` or `data.frame`), where each row represents an item and each column an attribute. Entries indicate whether an attribute is required (1) or not (0).
verbose	Logical; if `TRUE`, prints warning messages during checking process (e.g., insufficient items, missing patterns).

Details

The identifiability of the Q-matrix is essential for valid parameter estimation in CDMs. According to Gu & Xu (2021), identifiability can be categorized into:

Joint Strict Identifiability Ensures that both the Q-matrix and model parameters can be uniquely determined from the data, currently only applicable under DINA or DINO models. Requires three conditions:

- **Completeness (A)**: The Q-matrix contains a $K \times K$ identity submatrix (after row/column permutations).
- **Distinctness (B)**: All columns of the remaining part (excluding the identity block) are distinct.
- **Repetition (C)**: Each attribute appears in at least three items (i.e., column sums ≥ 3).

Joint Generic Identifiability (DINA/DINO) Means uniqueness holds for "almost all" parameter values (except on a set of measure zero). Applies when exactly one attribute has precisely two non-zero entries. The Q-matrix must have the structure: $\mathbf{Q} = \begin{pmatrix} 1 & \mathbf{0}^\top \\ 1 & \mathbf{v}^\top \\ \mathbf{0} & \mathbf{Q}^* \end{pmatrix}$. Then:

- If $\mathbf{v} = \mathbf{0}$, then $\mathbf{Q}^*$ must satisfy either: (i) Joint Strict Identifiability, or (ii) contain at least two identity submatrices $\rightarrow$ **Globally generically identifiable**.
- If $\mathbf{v} \neq \mathbf{0}, \mathbf{1}$, then $\mathbf{Q}^*$ must satisfy Joint Strict Identifiability $\rightarrow$ **Locally generically identifiable**.

Joint Generic Identifiability (Saturated Model) For general saturated models, requires:

- **Generic Completeness (D)**: Two different $K \times K$ submatrices exist, each having full rank and containing diagonal ones after permutation (indicating sufficient independent measurement of attributes).
- **Generic Repetition (E)**: At least one '1' exists outside these two submatrices.

This function first reconstructs the Q-matrix into standard form, then checks each condition accordingly.

Value

An object of class `"is.Qident"` — a list containing:

`Q.orig` Original input Q-matrix.

`Q.reconstructed` Reconstructed Q-matrix sorted by attribute pattern.

`arguments` A list containing all input arguments

`strictIdentifiability.obj` Results of checking Joint Strict Identifiability under DINA/DINO. Contains:

- `completeness`: TRUE if $K \times K$ identity submatrix exists.
- `distinctness`: TRUE if remaining columns are distinct.
- `repetition`: TRUE if every attribute appears more than 3 items.

All three must be TRUE for joint strict identifiability.

`genericIdentifiability.obj` Results for Joint Generic Identifiability under saturated models. Includes:

- `genericCompleteness`: TRUE if two different generic complete $K \times K$ submatrices exist.
- `genericRepetition`: TRUE if at least one '1' exists outside those submatrices.
- `Q1`, `Q2`: Identified generic complete submatrices (if found).
- `Q.star`: Remaining part after removing rows in `Q1` and `Q2`.

Both `genericCompleteness` and `genericRepetition` must be TRUE.

`genericIdentifiability.DINA.obj` Results for Joint Generic Identifiability under DINA/DINO. Includes:

- `locallyGenericIdentifiability`: TRUE if local generic identifiability holds.
- `globallyGenericIdentifiability`: TRUE if global generic identifiability holds.
- `Q.reconstructed.DINA`: Reconstructed Q-matrix with low-frequency attribute moved to first column.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

Gu, Y., & Xu, G. (2021). Sufficient and necessary conditions for the identifiability of the Q-matrix. *Statistica Sinica*, 31, 449–472. https://www.jstor.org/stable/26969691

See Also

`sim.Q`, `attributepattern`

Examples

```
library(Qval)
set.seed(123)

# Simulate a 5-attribute, 20-item Q-matrix
Q <- sim.Q(5, 20)

# Check identifiability
result <- is.Qident(Q, verbose = TRUE)
```

```
# View summary
print(result)
```

`plot` *Plot Methods for Various Qval Objects*

Description

Generate visualizations for objects created by the Qval package. The generic ‘plot‘ dispatches to appropriate methods based on object class:

CDM Barplot of attribute-pattern distribution (frequency and proportion).

sim.data Barplot of simulated attribute-pattern distribution (frequency and proportion).

validation Hull plot marking the suggested point in red (method = "Hull").

Usage

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

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

## S3 method for class 'validation'
plot(x, i = 1, ...)
```

Arguments

`x`	An object of class `CDM`, `sim.data`, or `validation`.
`...`	Additional arguments (currently unused).
`i`	For `validation` objects, the index of the item for which to plot the Hull curve.

Value

None. Functions are called for side effects (plotting).

Methods (by class)

- `plot(CDM)`: Plot method for `CDM` objects
- `plot(sim.data)`: Plot method for `sim.data` objects
- `plot(validation)`: Hull plot for `validation` objects

Examples

```
set.seed(123)
library(Qval)


K <- 4
I <- 20
IQ <- list(
  P0 = runif(I, 0.2, 0.4),
  P1 = runif(I, 0.6, 0.8)
)

#####################################################################
# Example 1: sim.data object                                        #
#####################################################################
Q <- sim.Q(K, I)
data.obj <- sim.data(Q = Q, N = 500, IQ = IQ,
                     model = "GDINA", distribute = "horder")

plot(data.obj)


#####################################################################
# Example 2: CDM object                                             #
#####################################################################
CDM.obj <- CDM(data.obj$dat, Q, model = "GDINA",
               method = "EM", maxitr = 2000, verbose = 1)
plot(CDM.obj)


#####################################################################
# Example 3: validation object (Hull plot)                          #
#####################################################################
MQ <- sim.MQ(Q, 0.1)

CDM.obj <- CDM(data.obj$dat, MQ)

############### ESA ###############
Hull.obj <- validation(data.obj$dat, MQ, CDM.obj,
                       method = "Hull", search.method = "ESA")

## plot Hull curve for item 20
plot(Hull.obj, 20)

############### PAA ###############
Hull.obj <- validation(data.obj$dat, MQ, CDM.obj,
                       method = "Hull", search.method = "PAA")

## plot Hull curve for item 20
plot(Hull.obj, 20)
```

print *Print Methods for Various Objects*

Description

Print concise, user-friendly summaries of objects generated by the Qval package. Supports objects of classes CDM, validation, sim.data, fit, is.Qident, att.hierarchy, as well as their corresponding summary objects.

Usage

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

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

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

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

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

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

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

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

## S3 method for class 'summary.sim.data'
print(x, ...)

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

## S3 method for class 'summary.is.Qident'
print(x, ...)

## S3 method for class 'summary.att.hierarchy'
print(x, ...)
```

Arguments

x	An object of the appropriate class (e.g., CDM, validation, sim.data, fit, is.Qident, att.hierarchy, or their summaries).
...	Currently unused. Additional arguments are ignored.

Details

The print methods provide an at-a-glance view of key information:

print.CDM displays sample size, item and attribute counts, and package information.

print.validation shows suggested modifications to the Q-matrix, marking changed entries with an asterisk.

print.sim.data reports dimensions of simulated data and offers guidance on extraction.

print.fit show basic fit indices.

print.is.Qident prints basic results from is.Qident.

print.att.hierarchy prints basic results from att.hierarchy.

print.summary.CDM prints fitted model details and alpha-pattern distribution from a summary.CDM object.

print.summary.validation prints suggested Q-matrix changes or a message if none are recommended.

print.summary.sim.data prints attribute-pattern frequencies and proportions from summary.sim.data.

print.summary.fit prints basic fit indices from summary.fit.

print.summary.is.Qident prints basic results from summary.is.Qident.

print.summary.att.hierarchy prints basic results from summary.att.hierarchy.

Value

Invisibly returns x.

Methods (by class)

- print(CDM): Print method for CDM objects
- print(validation): Print method for validation objects
- print(sim.data): Print method for sim.data objects
- print(fit): Print method for fit objects
- print(is.Qident): Print method for is.Qident objects
- print(att.hierarchy): Print method for att.hierarchy objects
- print(summary.CDM): Print method for summary.CDM objects
- print(summary.validation): Print method for summary.validation objects
- print(summary.sim.data): Print method for summary.sim.data objects
- print(summary.fit): Print method for summary.fit objects
- print(summary.is.Qident): Print method for summary.is.Qident objects
- print(summary.att.hierarchy): Print method for summary.att.hierarchy objects

Examples

```
set.seed(123)
library(Qval)


######################################################################
# Example 1: print a CDM object                                      #
######################################################################
Q <- sim.Q(3, 20)
IQ <- list(P0 = runif(20, 0.0, 0.2), P1 = runif(20, 0.8, 1.0))
data.obj <- sim.data(Q = Q, N = 500, IQ = IQ,
                     model = "GDINA", distribute = "horder")
CDM.obj <- CDM(data.obj$dat, Q, model = "GDINA",
               method = "EM", maxitr = 2000, verbose = 1)
print(CDM.obj)


######################################################################
# Example 2: print a validation object                               #
######################################################################
set.seed(123)
MQ <- sim.MQ(Q, 0.1)
CDM.obj <- CDM(data.obj$dat, MQ)
validation.obj <- validation(data.obj$dat, MQ, CDM.obj,
                             method = "GDI")
print(validation.obj)


######################################################################
# Example 3: print a sim.data object                                 #
######################################################################
set.seed(123)
Q2 <- sim.Q(3, 10)
data.obj2 <- sim.data(Q = Q2, N = 1000)
print(data.obj2)


######################################################################
# Example 4: print a fit object                                      #
######################################################################
set.seed(123)
Q2 <- sim.Q(3, 10)
fit.obj <- fit(Y = data.obj$dat, Q = Q, model = "GDINA")
print(fit.obj)


######################################################################
# Example 5: print summary objects                                   #
######################################################################
summary.CDM.obj <- summary(CDM.obj)
print(summary.CDM.obj)
```

```
summary.val.obj <- summary(validation.obj)
print(summary.val.obj)

summary.sim.obj <- summary(data.obj2)
print(summary.sim.obj)

summary.fit <- summary(fit.obj)
print(summary.fit)
```

`sim.data` *Generate Response*

Description

randomly generate response matrix according to certain conditions, including attributes distribution, item quality, sample size, Q-matrix and cognitive diagnosis models (CDMs).

Usage

```
sim.data(
  Q = NULL,
  N = NULL,
  IQ = list(P0 = NULL, P1 = NULL),
  att.str = NULL,
  model = "GDINA",
  distribute = "uniform",
  control = NULL,
  verbose = TRUE
)
```

Arguments

`Q`	The Q-matrix. A random 30×5 Q-matrix (`sim.Q`) will be used if `Q = NULL`.
`N`	Sample size. Default = 500.
`IQ`	A list containing two I-length vectors: `P0` and `P1`. `P0` represents the probability of examinees who have not mastered any attributes ($[00...0]$) correctly answering the item, while `P1` represents the probability of examinees who have mastered all attributes ($[11...1]$) correctly answering the item.
`att.str`	Specify attribute structures. `NULL`, by default, means there is no structure. Attribute structure needs be specified as a list - which will be internally handled by `att.structure` function. See examples. It can also be a matrix giving all permissible attribute profiles.
`model`	Type of model to be fitted; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"`, `"ACDM"`, `"LLM"`, or `"rRUM"`.

`distribute` Attribute distributions; can be `"uniform"` for the uniform distribution, `"mvnorm"` for the multivariate normal distribution (Chiu, Douglas, & Li, 2009) and `"horder"` for the higher-order distribution (Tu et al., 2022).

`control` A list of control parameters with elements:

- `sigma` A positive-definite symmetric matrix specifying the variance-covariance matrix when `distribute = "mvnorm"`. Default = 0.5 (Chiu, Douglas, & Li, 2009).
- `cutoffs` A vector giving the cutoff for each attribute when `distribute = "mvnorm"`. Default = $k/(1+K)$ (Chiu, Douglas, & Li, 2009).
- `theta` A vector of length N representing the higher-order ability for each examinee. By default, generate randomly from the standard normal distribution (Tu et al, 2022).
- `a` The slopes for the higher-order model when `distribute = "horder"`. Default = 1.5 (Tu et al, 2022).
- `b` The intercepts when `distribute = "horder"`. By default, select equally spaced values between -1.5 and 1.5 according to the number of attributes (Tu et al, 2022).
- `alpha` Used to generate a structured parameter distribution with a hierarchical structure when att.str is not NULL. This distribution is randomly drawn from a Dirichlet distribution, where alpha denotes the parameters of the Dirichlet distribution, and its length equals the number `L.str` of all valid attribute profiles $\boldsymbol{\alpha}$ under the hierarchical structure. Default by . `alpha = rep(1, L.str)`.

`verbose` Logical indicating to print information or not. Default is `TRUE`

Value

Object of class `sim.data`. An `sim.data` object initially gained by `simGDINA` function form `GDINA` package. Elements that can be extracted using method extract include:

`dat` An `N` × `I` simulated item response matrix.

`Q` The Q-matrix.

`attribute` An `N` × `K` matrix for inviduals' attribute patterns.

`catprob.parm` A list of non-zero success probabilities for each attribute mastery pattern.

`delta.parm` A list of delta parameters.

`higher.order.parm` Higher-order parameters.

`mvnorm.parm` Multivariate normal distribution parameters.

`LCprob.parm` A matrix of success probabilities for each attribute mastery pattern.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

Chiu, C.-Y., Douglas, J. A., & Li, X. (2009). Cluster Analysis for Cognitive Diagnosis: Theory and Applications. Psychometrika, 74(4), 633-665. DOI: 10.1007/s11336-009-9125-0.

Tu, D., Chiu, J., Ma, W., Wang, D., Cai, Y., & Ouyang, X. (2022). A multiple logistic regression-based (MLR-B) Q-matrix validation method for cognitive diagnosis models:A confirmatory approach. Behavior Research Methods. DOI: 10.3758/s13428-022-01880-x.

Examples

```
#####################################################################
#                           Example 1                               #
#          generate data follow the uniform distrbution             #
#####################################################################
library(Qval)

set.seed(123)

K <- 5
I <- 10
Q <- sim.Q(K, I)

IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)

data.obj <- sim.data(Q = Q, N = 100, IQ=IQ, model = "GDINA", distribute = "uniform")

print(data.obj$dat)

#####################################################################
#                           Example 2                               #
#          generate data follow the mvnorm distrbution              #
#####################################################################
set.seed(123)
K <- 5
I <- 10
Q <- sim.Q(K, I)

IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)

cutoffs <- sample(qnorm(c(1:K)/(K+1)), ncol(Q))
data.obj <- sim.data(Q = Q, N = 10, IQ=IQ, model = "GDINA", distribute = "mvnorm",
                 control = list(sigma = 0.5, cutoffs = cutoffs))

print(data.obj$dat)

#####################################################################
```

```
#                            Example 3                          #
#           generate data follow the horder distrbution         #
################################################################
set.seed(123)
K <- 5
I <- 10
Q <- sim.Q(K, I)

IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)

theta <- rnorm(10, 0, 1)
b <- seq(-1.5,1.5,length.out=K)
data.obj <- sim.data(Q = Q, N = 10, IQ=IQ, model = "GDINA", distribute = "horder",
                 control = list(theta = theta, a = 1.5, b = b))

print(data.obj$dat)
```

`sim.MQ` *Simulate Mis-specifications in Q-matrix*

Description

simulate certain `rate` mis-specifications in the Q-matrix.

Usage

```
sim.MQ(Q, rate, verbose = TRUE)
```

Arguments

`Q`	The Q-matrix (`sim.Q`) that need to simulate mis-specifications.
`rate`	The ratio of mis-specifications in the Q-matrix.
`verbose`	Logical indicating to print information or not. Default is `TRUE`

Value

An object of class `matrix`.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

Examples

```
library(Qval)

set.seed(123)

Q <- sim.Q(5, 10)
print(Q)

MQ <- sim.MQ(Q, 0.1)
print(MQ)
```

`sim.Q` *Generate a Random Q-matrix*

Description

generate a $I \times K$ Q-matrix randomly, which consisted of one-attribute q-vectors (50 This function ensures that the generated Q-matrix contains at least two identity matrices as a priority. Therefore, this function must also satisfy the condition that the number of items (I) must be at least twice the number of attributes (K).

Usage

```
sim.Q(K, I, att.str = NULL)
```

Arguments

`K`	The number of attributes of the Q-matrix.
`I`	The number of items.
`att.str`	Specify attribute structures. `NULL`, by default, means there is no structure. Attribute structure needs be specified as a list - which will be internally handled by `att.structure` function. See examples. It can also be a matrix giving all permissible attribute profiles.

Value

An object of class `matrix`.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

Najera, P., Sorrel, M. A., de la Torre, J., & Abad, F. J. (2021). Balancing fit and parsimony to improve Q-matrix validation. Br J Math Stat Psychol, 74 Suppl 1, 110-130. DOI: 10.1111/bmsp.12228.

Examples

```
library(Qval)

set.seed(123)

Q <- sim.Q(5, 10)
print(Q)
```

---

summary *Summary Methods for Various Objects*

Description

Generate concise summary statistics for objects created by the `Qval` package. The output is a named list tailored to the class of the input:

`CDM` contains the original call, dataset dimensions, model fit object, and attribute-pattern distribution.

`validation` contains the original call, suggested Q-matrix, and original Q-matrix.

`sim.data` contains the original call, dataset dimensions, and attribute-pattern distribution.

`fit` contains the original call, relative fit indices and absolute fit indices.

`is.Qident` contains the original call and results of whether the Q-matrix is identifiable.

`att.hierarchy` contains the results of iterative attribute hierarchy exploration.

Usage

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

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

## S3 method for class 'sim.data'
summary(object, ...)

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

## S3 method for class 'is.Qident'
summary(object, ...)

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

Arguments

object	An object of class CDM, validation, sim.data, fit, or is.Qident.
...	Currently unused. Additional arguments are ignored.

Value

A named list with class summary.<class> containing the components above.

Methods (by class)

- summary(CDM): Summary method for CDM objects
- summary(validation): Summary method for validation objects
- summary(sim.data): Summary method for sim.data objects
- summary(fit): Summary method for fit objects
- summary(is.Qident): Summary method for is.Qident objects
- summary(att.hierarchy): Summary method for att.hierarchy objects

Examples

```
set.seed(123)
library(Qval)


#####################################################################
# Example 1: summary a CDM object                                   #
#####################################################################
Q <- sim.Q(3, 20)
IQ <- list(P0 = runif(20, 0, 0.2), P1 = runif(20, 0.8, 1))
data.obj <- sim.data(Q, N = 500, IQ = IQ,
                     model = "GDINA", distribute = "horder")
CDM.obj <- CDM(data.obj$dat, Q, model = "GDINA", method = "EM")
summary(CDM.obj)


#####################################################################
# Example 2: summary a validation object                            #
#####################################################################
MQ <- sim.MQ(Q, 0.1)
CDM.obj2 <- CDM(data.obj$dat, MQ)
val.obj <- validation(data.obj$dat, MQ, CDM.obj2, method = "GDI")
summary(val.obj)


#####################################################################
# Example 3: summary a sim.data object                              #
#####################################################################
data.obj2 <- sim.data(Q = sim.Q(3, 10), N = 1000)
summary(data.obj2)
```

```
################################################################
# Example 4: summary a fit object                              #
################################################################
fit.obj <- fit(data.obj$dat, Q, model="GDINA")
summary(fit.obj)
```

---

update *Update Method for Various Objects*

Description

The update function provides a unified interface for refreshing or modifying existing analysis objects produced by the Qval package, including CDM, validation, sim.data, fit, is.Qident, att.hierarchy classes. By passing additional arguments, users can rerun fitting or simulation routines without reconstructing the entire object from scratch.

Usage

```
update(x, ...)

## S3 method for class 'CDM'
update(x, ...)

## S3 method for class 'validation'
update(x, ...)

## S3 method for class 'sim.data'
update(x, ...)

## S3 method for class 'fit'
update(x, ...)

## S3 method for class 'is.Qident'
update(x, ...)

## S3 method for class 'att.hierarchy'
update(x, ...)
```

Arguments

x	An object of class CDM, validation, sim.data, fit, is.Qident, att.hierarchy.
...	Additional arguments specific to the method being updated: • For CDM: Y, Q, model, method, mono.constraint, maxitr, verbose. • For validation: Y, Q, CDM.obj, par.method, mono.constraint, model, method, search.method, iter.level, maxitr, eps, alpha.level, criter, verbose.

- For sim.data: Q, N, IQ, model, distribute, control, verbose.
- For fit: Y, Q, model. ...

Details

The update methods internally extract the original call arguments from the input object, combine them with any new parameters provided in ..., and re-invoke the corresponding constructor (CDM, validation, sim.data, fit, is.Qident), att.hierarchy. This approach ensures consistency and preserves all untouched settings from the original object.

Value

An updated object of the same class as x, reflecting any changes in the supplied parameters.

Methods (by class)

- update(CDM): Update method for CDM objects
- update(validation): Update method for validation objects
- update(sim.data): Update method for sim.data objects
- update(fit): Update method for fit objects
- update(is.Qident): Update method for is.Qident objects
- update(att.hierarchy): Update method for att.hierarchy objects

Examples

```
set.seed(123)
library(Qval)


####################################################################
# Example 1: summary a CDM object                                  #
####################################################################
Q <- sim.Q(3, 20)
IQ <- list(P0 = runif(20, 0, 0.2), P1 = runif(20, 0.8, 1))
data.obj <- sim.data(Q, N = 500, IQ = IQ,
                     model = "GDINA", distribute = "horder")
CDM.obj <- CDM(data.obj$dat, Q, model = "GDINA", method = "BM")
summary(CDM.obj)

CDM.updated <- update(CDM.obj, method = "EM", maxitr = 1000)
summary(CDM.updated)


####################################################################
# Example 2: summary a validation object                           #
####################################################################
MQ <- sim.MQ(Q, 0.1)
CDM.obj2 <- CDM(data.obj$dat, MQ)
validation.obj <- validation(data.obj$dat, MQ, CDM.obj2,
                             method = "GDI")
```

```
summary(validation.obj)

validation.updated <- update(validation.obj, method = "Hull")
summary(validation.updated)


#################################################################
# Example 3: summary a sim.data object                          #
#################################################################
data.obj2 <- sim.data(Q = sim.Q(3, 10), N = 1000)
summary(data.obj2)

data.updated <- update(data.obj2, N = 200)
summary(data.updated)


#################################################################
# Example 4: summary a fit object                               #
#################################################################
fit.obj <- fit(data.obj$dat, Q, model = "GDINA")
summary(fit.obj)

data.updated <- update(fit.obj, model = "DINA")
summary(data.updated)
```

`validation` *Perform Q-matrix Validation Methods*

Description

This function uses generalized Q-matrix validation methods to validate the Q-matrix, including commonly used methods such as GDI (de la Torre, & Chiu, 2016; Najera, Sorrel, & Abad, 2019; Najera et al., 2020), Wald (Ma, & de la Torre, 2020), Hull (Najera et al., 2021), and MLR-B (Tu et al., 2022). It supports different iteration methods (test level or item level; Najera et al., 2020; Najera et al., 2021; Tu et al., 2022) and can apply various attribute search methods (ESA, SSA, PAA; de la Torre, 2008; Terzi, & de la Torre, 2018; Qin & Guo, 2025).

Usage

```
validation(
  Y,
  Q,
  CDM.obj = NULL,
  par.method = "EM",
  mono.constraint = FALSE,
  model = "GDINA",
  method = "GDI",
```

```
  search.method = "PAA",
  iter.level = "no",
  maxitr = 1,
  eps = 0.95,
  alpha.level = NULL,
  criter = NULL,
  verbose = TRUE
)
```

Arguments

Y	A required $N \times I$ matrix or data.frame consisting of the responses of N individuals to $N \times I$ items. Missing values need to be coded as NA.
Q	A required binary $I \times K$ matrix containing the attributes not required or required master the items. The ith row of the matrix is a binary indicator vector indicating which attributes are not required (coded by 0) and which attributes are required (coded by 1) to master item i.
CDM.obj	An object of class CDM.obj. When it is not NULL, it enables rapid validation of the Q-matrix without the need for parameter estimation. @seealso CDM.
par.method	Type of method to estimate CDMs' parameters; one out of "EM", "BM". Default = "EM". However, "BM" is only available when method = "GDINA".
mono.constraint	Logical indicating whether monotonicity constraints should be fulfilled in estimation. Default = FALSE.
model	Type of model to fit; can be "GDINA", "LCDM", "DINA", "DINO" , "ACDM", "LLM", or "rRUM". Default = "GDINA". @seealso CDM.
method	The methods to validata Q-matrix, can be "GDI", "Wald", "Hull", "MLR-B" and "beta". The "model" must be "GDINA" when method = "Wald". Please note that the β method has different meanings when applying different search algorithms. For more details, see section 'Search algorithm' below. Default = "GDI". See details.
search.method	Character string specifying the search method to use during validation. "ESA" for exhaustive search algorithm. Cannot be used with the "Wald" method. "SSA" for sequential search algorithm (see de la Torre, 2008; Terzi & de la Torre, 2018). It will be equal to "forward" when method = "Wald". "PAA" for priority attribute algorithm. "stepwise" only for the "Wald" method "beta" only for the "beta" method
iter.level	Can be "no", "item" level, "test.att" or "test" level. Default = "no" and "test.att" can not for "Wald" and "MLR-B". See details.
maxitr	Number of max iterations. Default = 1.
eps	Cut-off points of $PVAF$, will work when the method is "GDI" or "Wald". Default = 0.95. When eps = 'logit', the predicted eps by Najera et al. (2019) will be used. See details.
alpha.level	alpha level for the wald test. Default = 0.05 for "Wald" and 0.01 for "MLR-B"

`criter` The kind of fit-index value. When `method = "Hull"`, it can be `R2` for $R^2_{McFadden}$ @seealso `get.R2` or `'PVAF'` for the proportion of variance accounted for ($PVAF$) @seealso `get.PVAF`. When `method = "beta"`, it can be `'AIC'`, `'BIC'`, `'CAIC'` or `'SABIC'`. Default = `"PVAF"` for `'Hull'` and default = `"AIC"` for `'beta'`. See details.

`verbose` Logical indicating to print iterative information or not. Default is `TRUE`

Value

An object of class `validation` containing the following components:

`Q.orig` The original Q-matrix that maybe contain some mis-specifications and need to be validated.

`Q.sug` The Q-matrix that suggested by certain validation method.

`time.cost` The time cost to finish the function.

`process` A matrix that contains the modification process of each question during each iteration. Each row represents an iteration, and each column corresponds to the q-vector index of the respective question. The order of the indices is consistent with the row numbering in the matrix generated by the `attributepattern` function in the `GDINA` package. Only when `maxitr` > 1, the value is available.

`iter` The number of iteration. Only when `maxitr` > 1, the value is available.

`priority` An `I` × `K` matrix that contains the priority of every attribute for each item. Only when the `search.method` is `"PAA"`, the value is available. See details.

`Hull.fit` A `list` containing all the information needed to plot the Hull plot, which is available only when `method = "Hull"`.

`arguments` A list containing all input arguments

The GDI method

The GDI method (de la Torre & Chiu, 2016), as the first Q-matrix validation method applicable to saturated models, serves as an important foundation for various mainstream Q-matrix validation methods.

The method calculates the proportion of variance accounted for ($PVAF$; @seealso `get.PVAF`) for all possible q-vectors for each item, selects the q-vector with a $PVAF$ just greater than the cut-off point (or Epsilon, EPS) as the correction result, and the variance ζ^2 is the generalized discriminating index (GDI; de la Torre & Chiu, 2016). Therefore, the GDI method is also considered as a generalized extension of the $delta$ method (de la Torre, 2008), which also takes maximizing discrimination as its basic idea. In the GDI method, ζ^2 is defined as the weighted variance of the correct response probabilities across all mastery patterns, that is:

$$\zeta^2 = \sum_{l=1}^{2^K} \pi_l \left[P(X_{pi} = 1|\boldsymbol{\alpha}_l) - \bar{P}_i\right]^2$$

where π_l represents the prior probability of mastery pattern l; $\bar{P}_i = \sum_{k=1}^{K} \pi_l P(X_{pi} = 1|\boldsymbol{\alpha}_l)$ is the weighted average of the correct response probabilities across all attribute mastery patterns. When the q-vector is correctly specified, the calculated ζ^2 should be maximized, indicating the

maximum discrimination of the item. However, in reality, ζ^2 continues to increase when the q-vector is over-specified, and the more attributes that are over-specified, the larger ζ^2 becomes. The q-vector with all attributes set to 1 (i.e., $\boldsymbol{q}_{1:K}$) has the largest ζ^2 (de la Torre, 2016). This is because an increase in attributes in the q-vector leads to an increase in item parameters, resulting in greater differences in correct response probabilities across attribute patterns and, consequently, increased variance. However, this increase in variance is spurious. Therefore, de la Torre et al. calculated $PVAF = \frac{\zeta^2}{\zeta^2_{1:K}}$ to describe the degree to which the discrimination of the current q-vector explains the maximum discrimination. They selected an appropriate $PVAF$ cut-off point to achieve a balance between q-vector fit and parsimony. According to previous studies, the $PVAF$ cut-off point is typically set at 0.95 (Ma & de la Torre, 2020; Najera et al., 2021). Najera et al. (2019; 2020) proposed using multinomial logistic regression to predict a more appropriate cut-off point for $PVAF$. The cut-off point is denoted as eps, and the predicted regression equation is as follows:

$$\log\left(\frac{eps}{1-eps}\right) = \text{logit}(eps) = -0.405 + 2.867 \cdot IQ + 4.840 \times 10^{-4} \cdot N - 3.316 \times 10^{-3} \cdot I$$

Where IQ represents the item quality, calculated as the average difference between the probability of examinee with all attributes answering the item correctly and the probability of an examinee with no attributes answering the question correctly ($IQ = \frac{1}{I}\sum_{i=1}^{I}\left[P_i\left(\mathbf{1}\right) - P_i\left(\mathbf{0}\right)\right]$), and N and I represent the number of examinees and the number of questions, respectively.

The Wald method

The Wald method (Ma & de la Torre, 2020) combines the Wald test with $PVAF$ to correct the Q-matrix at the item level. Its basic logic is as follows: when correcting item i, the single attribute that maximizes the $PVAF$ value is added to a vector with all attributes set to $\mathbf{0}$ (i.e., $\boldsymbol{q} = (0, 0, \ldots, 0)$) as a starting point. In subsequent iterations, attributes in this vector are continuously added or removed through the Wald test. The correction process ends when the $PVAF$ exceeds the cut-off point or when no further attribute changes occur. The Wald statistic follows an asymptotic χ^2 distribution with a degree of freedom of $2^{K^*} - 1$.

The calculation method is as follows:

$$Wald = \left[\boldsymbol{R} \times \boldsymbol{P}_i(\boldsymbol{\alpha})\right]' \left(\boldsymbol{R} \times \boldsymbol{V}_i \times \boldsymbol{R}\right)^{-1} \left[\boldsymbol{R} \times P_i(\boldsymbol{\alpha})\right]$$

$\boldsymbol{R}$ represents the restriction matrix (@seealso `get.Rmatrix`); $\boldsymbol{P}_i(\boldsymbol{\alpha})$ denotes the vector of correct response probabilities for item i; $\boldsymbol{V}_i$ is the variance-covariance matrix of the correct response probabilities for item i, which can be obtained by multiplying the $\boldsymbol{M}_i$ matrix (de la Torre, 2011) with the variance-covariance matrix of item parameters $\boldsymbol{\Sigma}_i$, i.e., $\boldsymbol{V}_i = \boldsymbol{M}_i \times \boldsymbol{\Sigma}_i$. The $\boldsymbol{\Sigma}_i$ can be derived by inverting the information matrix. Using the empirical cross-product information matrix (de la Torre, 2011) to calculate $\boldsymbol{\Sigma}_i$.

$\boldsymbol{M}_i$ is a $2^{K^*} \times 2^{K^*}$ matrix (@seealso `get.Mmatrix`) that represents the relationship between the parameters of item i and the attribute mastery patterns. The rows represent different mastery patterns, while the columns represent different item parameters.

In Package Qval, parLapply will be used to accelerate the Wald method.

The Hull method

The Hull method (Najera et al., 2021) addresses the issue of the cut-off point in the GDI method and demonstrates good performance in simulation studies. Najera et al. applied the Hull method for determining the number of factors to retain in exploratory factor analysis (Lorenzo-Seva et al., 2011) to the retention of attribute quantities in the q-vector, specifically for Q-matrix validation. The Hull method aligns with the GDI approach in its philosophy of seeking a balance between fit and parsimony. While GDI relies on a preset, arbitrary cut-off point to determine this balance, the Hull method utilizes the most pronounced elbow in the Hull plot to make this judgment. The most pronounced elbow is determined using the following formula:

$$st = \frac{(f_k - f_{k-1})/(np_k - np_{k-1})}{(f_{k+1} - f_k)/(np_{k+1} - np_k)}$$

where f_k represents the fit-index value (can be $PVAF$ @seealso `get.PVAF` or $R2$ @seealso `get.R2`) when the q-vector contains k attributes, similarly, f_{k-1} and f_{k+1} represent the fit-index value when the q-vector contains $k-1$ and $k+1$ attributes, respectively. np_k denotes the number of parameters when the q-vector has k attributes, which is 2^k for a saturated model. Likewise, np_{k-1} and np_{k+1} represent the number of parameters when the q-vector has $k-1$ and $k+1$ attributes, respectively. The Hull method calculates the st index for all possible q-vectors and retains the q-vector with the maximum st index as the corrected result. Najera et al. (2021) removed any concave points from the Hull plot, and when only the first and last points remained in the plot, the saturated q-vector was selected.

The MLR-B method

The MLR-B method proposed by Tu et al. (2022) differs from the GDI, Wald and Hull method in that it does not employ $PVAF$. Instead, it directly uses the marginal probabilities of attribute mastery for examinees to perform multivariate logistic regression on their observed scores. This approach assumes all possible q-vectors and conducts $2^K - 1$ regression modelings. After proposing regression equations that exclude any insignificant regression coefficients, it selects the q-vector corresponding to the equation with the minimum AIC value as the validation result. The performance of this method in both the LCDM and GDM models even surpasses that of the Hull method (Tu et al., 2022), making it an efficient and reliable approach for Q-matrix validation.

In Package Qval, parLapply will be used to accelerate the MLR-B method when `search.method` is not `"PAA"`.

The β method

The β method (Li & Chen, 2024) addresses the Q-matrix validation problem from the perspective of signal detection theory. Signal detection theory posits that any stimulus is a signal embedded in noise, where the signal always overlaps with noise. The β method treats the correct q-vector as the signal and other possible q-vectors as noise. The goal is to identify the signal from the noise, i.e., to correctly identify the q-vector. For item i with the q-vector of the c-th type, the β index is computed as follows:

$$\beta_{ic} = \sum_{l=1}^{2^K} \left| \frac{r_{li}}{n_l} P_{ic}(\boldsymbol{\alpha_l}) - \left(1 - \frac{r_{li}}{n_l}\right) [1 - P_{ic}(\boldsymbol{\alpha_l})] \right| = \sum_{l=1}^{2^K} \left| \frac{r_{li}}{n_l} - [1 - P_{ic}(\boldsymbol{\alpha_l})] \right|$$

In the formula, r_{li} represents the number of examinees in knowledge state l who correctly answered item i, while n_l is the total number of examinees in knowledge state l. $P_{ic}(\boldsymbol{\alpha_l})$ denotes the probability that an examinee in knowledge state l answers item i correctly when the q-vector for item i is of the c-th type. In fact, $\frac{r_{li}}{n_l}$ is the observed probability that an examinee in knowledge state l answers item i correctly, and β_{jc} represents the difference between the actual proportion of correct answers for item i in each knowledge state and the expected probability of answering the item incorrectly in that state. Therefore, to some extent, β_{jc} can be considered as a measure of discriminability, and the β method posits that the correct q-vector maximizes β_{jc}, i.e.:

$$\boldsymbol{q}_i = \arg\max_{\boldsymbol{q}} \left(\beta_{jc} : \boldsymbol{q} \in \left\{\boldsymbol{q}_{ic},\, c = 1, 2, \ldots, 2^K - 1\right\}\right)$$

Therefore, essentially, β_{jc} is an index similar to GDI. Both increase as the number of attributes in the q-vector increases. Unlike the GDI method, the β method does not continue to compute $\beta_{jc}/\beta_{j[11\ldots1]}$ but instead uses the minimum AIC value to determine whether the attributes in the q-vector are sufficient. In Package Qval, parLapply will be used to accelerate the β method.

Please note that the β method has different meanings when applying different search algorithms. For more details, see section 'Search algorithm' below.

Iterative procedure

The iterative procedure that one item modification at a time is item level iteration (`iter.level = "item"`) in (Najera et al., 2020, 2021). The steps of the `item` level iterative procedure algorithm are as follows:

Step1 Fit the `CDM` according to the item responses and the provisional Q-matrix ($\boldsymbol{Q}^0$).

Step2 Validate the provisional Q-matrix and gain a suggested Q-matrix ($\boldsymbol{Q}^1$).

Step3 for each item, $PVAF_{0i}$ as the $PVAF$ of the provisional q-vector specified in $\boldsymbol{Q}^0$, and $PVAF_{1i}$ as the $PVAF$ of the suggested q-vector in $\boldsymbol{Q}^1$.

Step4 Calculate all items' $\Delta PVAF_i$, defined as $\Delta PVAF_i = |PVAF_{1i} - PVAF_{0i}|$

Step5 Define the hit item as the item with the highest $\Delta PVAF_i$.

Step6 Update $\boldsymbol{Q}^0$ by changing the provisional q-vector by the suggested q-vector of the hit item.

Step7 Iterate over Steps 1 to 6 until $\sum_{i=1}^{I} \Delta PVAF_i = 0$

When the Q-matrix validation method is `"MLR-B"` or `"Hull"` when `criter = "AIC"` or `criter = "R2"`, $PVAF$ is not used. In this case, the criterion for determining which item's index will be replaced is AIC or R^2, respectively.

The iterative procedure that the entire Q-matrix is modified at each iteration is test level iteration (`iter.level = "test"`) (Najera et al., 2020; Tu et al., 2022). The steps of the `test` level iterative procedure algorithm are as follows:

Step1 Fit the `CDM` according to the item responses and the provisional Q-matrix ($\boldsymbol{Q}^0$).

Step2 Validate the provisional Q-matrix and gain a suggested Q-matrix ($\boldsymbol{Q}^1$).

Step3 Check whether $\boldsymbol{Q}^1 = \boldsymbol{Q}^0$. If `TRUE`, terminate the iterative algorithm. If `FALSE`, Update $\boldsymbol{Q}^0$ as $\boldsymbol{Q}^1$.

Step4 Iterate over Steps 1 and 3 until one of conditions as follows is satisfied: 1. $\boldsymbol{Q}^1 = \boldsymbol{Q}^0$; 2. Reach the maximum number of iterations (`maxitr`); 3. $\boldsymbol{Q}^1$ does not satisfy the condition that an attribute is measured by one item at least.

`iter.level = 'test.att'` will use a method called the test-attribute iterative procedure (Najera et al., 2021), which modifies all items in each iteration while following the principle of minimizing changes in the number of attributes. Therefore, the test-attribute iterative procedure and the test-level iterative procedure follow the same process for large items. The key difference is that the test-attribute iterative procedure only allows minimal adjustments to the q-vector in each iteration. For example, if the original q-vector is $[0010]$ and the validation methods suggest $[1110]$, the test-level iterative procedure can directly update the q-vector to $[1110]$. In contrast, the test-attribute iterative procedure can only make a gradual adjustment, first modifying the q-vector to either $[1010]$ or $[0110]$. As a result, the test-attribute iterative procedure is more cautious than the test-level iterative procedure and may require more iterations.

Search algorithm

Three search algorithms are available: Exhaustive Search Algorithm (ESA), Sequential Search Algorithm (SSA), and Priority Attribute Algorithm (PAA). ESA is a brute-force algorithm. When validating the q-vector of a particular item, it traverses all possible q-vectors and selects the most appropriate one based on the chosen Q-matrix validation method. Since there are 2^{K-1} possible q-vectors with K attributes, ESA requires 2^{K-1} searches for each item.

SSA reduces the number of searches by adding one attribute at a time to the q-vector in a stepwise manner. Therefore, in the worst-case scenario, SSA requires $K(K-1)/2$ searches. The detailed steps are as follows:

Step 1 Define an empty q-vector $\boldsymbol{q}^0 = [00...0]$ of length K, where all elements are 0.

Step 2 Examine all single-attribute q-vectors, which are those formed by changing one of the 0s in $\boldsymbol{q}^0$ to 1. According to the criteria of the chosen Q-matrix validation method, select the optimal single-attribute q-vector, denoted as $\boldsymbol{q}^1$.

Step 3 Examine all two-attribute q-vectors, which are those formed by changing one of the 0s in $\boldsymbol{q}^1$ to 1. According to the criteria of the chosen Q-matrix validation method, select the optimal two-attribute q-vector, denoted as $\boldsymbol{q}^2$.

Step 4 Repeat this process until $\boldsymbol{q}^K$ is found, or the stopping criterion of the chosen Q-matrix validation method is met.

PAA is a highly efficient and concise algorithm that evaluates whether each attribute needs to be included in the q-vector based on the priority of the attributes. @seealso `get.priority`. Therefore, even in the worst-case scenario, PAA only requires K searches. The detailed process is as follows:

Step 1 Using the applicable CDM (e.g. the G-DINA model) to estimate the model parameters and obtain the marginal attribute mastery probabilities matrix $\boldsymbol{\Lambda}$

Step 2 Use LASSO regression to calculate the priority of each attribute in the q-vector for item i

Step 3 Check whether each attribute is included in the optimal q-vector based on the attribute priorities from high to low seriatim and output the final suggested q-vector according to the criteria of the chosen Q-matrix validation method.

The calculation of priorities is straightforward (Qin & Guo, 2025): the priority of an attribute is the regression coefficient obtained from a LASSO multinomial logistic regression, with the attribute as the independent variable and the response data from the examinees as the dependent variable. The formula (Tu et al., 2022) is as follows:

$$\log[\frac{P(X_{pi}=1|\mathbf{\Lambda}_p)}{P(X_{pi}=0|\mathbf{\Lambda}_p)}] = logit[P(X_{pi}=1|\mathbf{\Lambda}_p)] = \beta_{i0} + \beta_{i1}\Lambda_{p1} + \ldots + \beta_{ik}\Lambda_{pk} + \ldots + \beta_{iK}\Lambda_{pK}$$

Where X_{pi} represents the response of examinee p on item i, $\mathbf{\Lambda}_p$ denotes the marginal mastery probabilities of examinee p (which can be obtained from the return value `alpha.P` of the `CDM` function), β_{i0} is the intercept term, and β_{ik} represents the regression coefficient.

The LASSO loss function can be expressed as:

$$l_{lasso}(\boldsymbol{X}_i|\mathbf{\Lambda}) = l(\boldsymbol{X}_i|\mathbf{\Lambda}) - \lambda|\boldsymbol{\beta}_i|$$

Where $l_{lasso}(\boldsymbol{X}_i|\mathbf{\Lambda})$ is the penalized likelihood, $l(\boldsymbol{X}_i|\mathbf{\Lambda})$ is the original likelihood, and λ is the tuning parameter for penalization (a larger value imposes a stronger penalty on $\boldsymbol{\beta}_i = [\beta_{i1}, \ldots, \beta_{ik}, \ldots, \beta_{iK}]$). The priority for attribute i is defined as: $\boldsymbol{priority}_i = \boldsymbol{\beta}_i = [\beta_{i1}, \ldots, \beta_{ik}, \ldots, \beta_{iK}]$

It should be noted that the Wald method proposed by Ma and de la Torre (2020) uses a `"stepwise"` search approach. This approach involves incrementally adding or removing 1 from the q-vector and evaluating the significance of the change using the Wald test: 1. If removing a 1 results in non-significance (indicating that the 1 is unnecessary), the 1 is removed from the q-vector; otherwise, the q-vector remains unchanged. 2. If adding a 1 results in significance (indicating that the 1 is necessary), the 1 is added to the q-vector; otherwise, the q-vector remains unchanged. The process stops when the q-vector no longer changes or when the PVAF reaches the preset cut-off point (i.e., 0.95). Stepwise are unique search approach of the Wald method, and users should be aware of this. Since stepwise is inefficient and differs significantly from the extremely high efficiency of PAA, `Qval` package also provides `PAA` for q-vector search in the Wald method. When applying the PAA version of the Wald method, the search still examines whether each attribute is necessary (by checking if the Wald test reaches significance after adding the attribute) according to attribute priority. The search stops when no further necessary attributes are found or when the PVAF reaches the preset cut-off point (i.e., 0.95). The "forward" search approach is another search method available for the Wald method, which is equivalent to `"SSA"`. When `"Wald"` uses `search.method = "SSA"`, it means that the Wald method is employing the forward search approach. Its basic process is the same as `'stepwise'`, except that it does not remove elements from the q-vector. Therefore, the "forward" search approach is essentially equivalent to SSA.

Please note that, since the β method essentially selects q-vectors based on AIC, even without using the iterative process, the β method requires multiple parameter estimations to obtain the AIC values for different q-vectors. Therefore, the β method is more time-consuming and computationally intensive compared to the other methods. Li and Chen (2024) introduced a specialized search approach for the β method, which is referred to as the β search (`search.method = 'beta'`). The number of searches required is $2^{K-2} + K + 1$, and the specific steps are as follows:

Step 1 For item i, sequentially examine the β values for each single-attribute q-vector, select the largest β_{most} and the smallest β_{least}, along with the corresponding attributes k_{most} and k_{least}. (K searches)

Step 2 Then, add all possible q-vectors (a total of $2^K - 1$) containing attribute k_{most} and not containing k_{least} to the search space $\boldsymbol{S}_i$ (a total of 2^{K-2})), and unconditionally add the saturated q-vector $[11\ldots1]$ to $\boldsymbol{S}_i$ to ensure that it is tested.

Step 3 Select the q-vector with the minimum AIC from $\boldsymbol{S}_i$ as the final output of the β method. (The remaining $2^{K-2} + 1$ searches)

The Qval package also provides three search methods, ESA, SSA, and PAA, for the β method. When the β method applies these three search methods, Q-matrix validation can be completed without calculating any β values, as the β method essentially uses AIC for selecting q-vectors. For example, when applying ESA, the β method does not need to perform Step 1 of the β search and only needs to include all possible q-vectors (a total of $2^K - 1$) in $\boldsymbol{S}_i$, then outputs the corresponding q-vector based on the minimum AIC. When applying SSA or PAA, the β method also does not require any calculation of β values. In this case, the β method is consistent with the Q-matrix validation process described by Chen et al. (2013) using relative fit indices. Therefore, when the β method does not use β search, it is equivalent to the method of Chen et al. (2013). To better implement Chen et al. (2013)'s Q-matrix validation method using relative fit indices, the Qval package also provides BIC, $CAIC$, and $SABIC$ as alternatives to validate q-vectors, in addition to AIC.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

Chen, J., de la Torre, J., & Zhang, Z. (2013). Relative and Absolute Fit Evaluation in Cognitive Diagnosis Modeling. Journal of Educational Measurement, 50(2), 123-140. DOI: 10.1111/j.1745-3984.2012.00185.x

de la Torre, J., & Chiu, C. Y. (2016). A General Method of Empirical Q-matrix Validation. Psychometrika, 81(2), 253-273. DOI: 10.1007/s11336-015-9467-8.

de la Torre, J. (2008). An Empirically Based Method of Q-Matrix Validation for the DINA Model: Development and Applications. Journal of Education Measurement, 45(4), 343-362. DOI: 10.1111/j.1745-3984.2008.00069.x.

Li, J., & Chen, P. (2024). A new Q-matrix validation method based on signal detection theory. British Journal of Mathematical and Statistical Psychology, 00, 1–33. DOI: 10.1111/bmsp.12371

Lorenzo-Seva, U., Timmerman, M. E., & Kiers, H. A. (2011). The Hull method for selecting the number of common factors. Multivariate Behavioral Research, 46, 340–364. DOI: 10.1080/00273171.2011.564527.

Ma, W., & de la Torre, J. (2020). An empirical Q-matrix validation method for the sequential generalized DINA model. British Journal of Mathematical and Statistical Psychology, 73(1), 142-163. DOI: 10.1111/bmsp.12156.

McFadden, D. (1974). Conditional logit analysis of qualitative choice behavior. In P. Zarembka (Ed.), Frontiers in economics (pp. 105–142). New York, NY: Academic Press.

Najera, P., Sorrel, M. A., & Abad, F. J. (2019). Reconsidering Cutoff Points in the General Method of Empirical Q-Matrix Validation. Educational and Psychological Measurement, 79(4), 727-753. DOI: 10.1177/0013164418822700.

Najera, P., Sorrel, M. A., de la Torre, J., & Abad, F. J. (2020). Improving Robustness in Q-Matrix Validation Using an Iterative and Dynamic Procedure. Applied Psychological Measurement, 44(6), 431-446. DOI: 10.1177/0146621620909904.

Najera, P., Sorrel, M. A., de la Torre, J., & Abad, F. J. (2021). Balancing fit and parsimony to improve Q-matrix validation. British Journal of Mathematical and Statistical Psychology, 74 Suppl 1, 110-130. DOI: 10.1111/bmsp.12228.

Qin, H., & Guo, L. (2025). Priority attribute algorithm for Q-matrix validation: A didactic. Behavior Research Methods, 57(1), 31. DOI: 10.3758/s13428-024-02547-5.

Terzi, R., & de la Torre, J. (2018). An Iterative Method for Empirically-Based Q-Matrix Validation. International Journal of Assessment Tools in Education, 248-262. DOI: 10.21449/ijate.40719.

Tu, D., Chiu, J., Ma, W., Wang, D., Cai, Y., & Ouyang, X. (2022). A multiple logistic regression-based (MLR-B) Q-matrix validation method for cognitive diagnosis models: A confirmatory approach. Behavior Research Methods. DOI: 10.3758/s13428-022-01880-x.

Examples

```
#####################################################################
#                           Example 1                               #
#             The GDI method to validate Q-matrix                   #
#####################################################################
set.seed(123)

library(Qval)

## generate Q-matrix and data
K <- 3
I <- 20
Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
data <- sim.data(Q = Q, N = 500, IQ = IQ,
                         model = "GDINA", distribute = "horder")

## simulate random mis-specifications
MQ <- sim.MQ(Q, 0.2)

## using MMLE/EM to fit CDM model first
CDM.obj <- CDM(data$dat, MQ)

## using the fitted CDM.obj to avoid extra parameter estimation.
Q.GDI.obj <- validation(data$dat, MQ, CDM.obj=CDM.obj, method = "GDI")
## check QRR
print(zQRR(Q, Q.GDI.obj$Q.sug))


## also can validate the Q-matrix directly
Q.GDI.obj <- validation(data$dat, MQ)

## item level iteration
Q.GDI.obj <- validation(data$dat, MQ, method = "GDI",
                        iter.level = "item", maxitr = 150)

## search method
Q.GDI.obj <- validation(data$dat, MQ, method = "GDI",
                        search.method = "ESA")
```

```
## cut-off point
Q.GDI.obj <- validation(data$dat, MQ, method = "GDI",
                        eps = 0.90)

## check QRR
print(zQRR(Q, Q.GDI.obj$Q.sug))




#####################################################################
#                            Example 2                              #
#             The Hull method to validate Q-matrix                  #
#####################################################################

set.seed(123)

library(Qval)

## generate Q-matrix and data
K <- 4
I <- 20
Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
data <- sim.data(Q = Q, N = 500, IQ = IQ, model = "GDINA",
                         distribute = "horder")

## simulate random mis-specifications
MQ <- sim.MQ(Q, 0.1)

## using MMLE/EM to fit CDM first
CDM.obj <- CDM(data$dat, MQ)

## using the fitted CDM.obj to avoid extra parameter estimation.
Q.Hull.obj <- validation(data$dat, MQ, CDM.obj, method = "Hull")


## also can validate the Q-matrix directly
Q.Hull.obj <- validation(data$dat, MQ, method = "Hull")

## change PVAF to R2 as fit-index
Q.Hull.obj <- validation(data$dat, MQ, method = "Hull", criter = "R2")

## check QRR
print(zQRR(Q, Q.Hull.obj$Q.sug))
```

```
#####################################################################
#                            Example 3                              #
#             The MLR-B method to validate Q-matrix                 #
#####################################################################

set.seed(123)

library(Qval)

## generate Q-matrix and data
K <- 4
I <- 20
Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
data <- sim.data(Q = Q, N = 500, IQ = IQ, model = "GDINA",
                         distribute = "horder")

## simulate random mis-specifications
MQ <- sim.MQ(Q, 0.1)

## using MMLE/EM to fit CDM first
CDM.obj <- CDM(data$dat, MQ)

## using the fitted CDM.obj to avoid extra parameter estimation.
Q.MLR.obj <- validation(data$dat, MQ, CDM.obj, method = "MLR-B")

## check QRR
print(zQRR(Q, Q.MLR.obj$Q.sug))


## also can validate the Q-matrix directly
Q.MLR.obj <- validation(data$dat, MQ, method  = "MLR-B")

## check QRR
print(zQRR(Q, Q.MLR.obj$Q.sug))
```

`Wald.test` *Wald Test for Two Q-vectors*

Description

This function flexibly provides the Wald test for any two q-vectors of a given item in the Q-matrix, but requires that the two q-vectors differ by only one attribute. Additionally, this function needs to accept a `CDM.obj`.

Usage

```
Wald.test(CDM.obj, q1, q2, i = 1)
```

Arguments

CDM.obj	An object of class CDM.obj. @seealso CDM.
q1	A q-vector
q2	Another q-vector
i	the item needed to be validated

Details

$$Wald = [\boldsymbol{R} \times \boldsymbol{P}_i(\boldsymbol{\alpha})]^{'} (\boldsymbol{R} \times \boldsymbol{V}_i \times \boldsymbol{R})^{-1} [\boldsymbol{R} \times P_i(\boldsymbol{\alpha})]$$

Value

An object of class htest containing the following components:

statistic The statistic of the Wald test.

parameter the degrees of freedom for the Wald-statistic.

p.value The p value

Examples

```
library(Qval)
set.seed(123)

K <- 3
I <- 20
N <- 500
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
Q <- sim.Q(K, I)
data <- sim.data(Q = Q, N = N, IQ = IQ, model = "GDINA", distribute = "horder")

CDM.obj <- CDM(data$dat, Q)

q1 <- c(1, 0, 0)
q2 <- c(1, 1, 0)

## Discuss whether there is a significant difference when
## the q-vector of the 2nd item in the Q-matrix is q1 or q2.
Wald.test.obj <- Wald.test(CDM.obj, q1, q2, i=2)

print(Wald.test.obj)
```

zOSR *Calculate Over-Specification Rate (OSR)*

Description

Calculate Over-Specification Rate (OSR)

Usage

```
zOSR(Q.true, Q.sug)
```

Arguments

Q.true	The true Q-matrix.
Q.sug	The Q-matrix that has been validated.

Details

The OSR is defined as:

$$OSR = \frac{\sum_{i=1}^{I} \sum_{k=1}^{K} I(q_{ik}^{t} < q_{ik}^{s})}{IK}$$

where q_{ik}^{t} denotes the kth attribute of item i in the true Q-matrix (Q.true), q_{ik}^{s} denotes kth attribute of item i in the suggested Q-matrix(Q.sug), and $I(\cdot)$ is the indicator function.

Value

A numeric (OSR index).

Examples

```
library(Qval)

set.seed(123)

Q1 <- sim.Q(5, 30)
Q2 <- sim.MQ(Q1, 0.1)
OSR <- zOSR(Q1, Q2)
print(OSR)
```

zQRR *Calculate Q-matrix Recovery Rate (QRR)*

Description

Calculate Q-matrix Recovery Rate (QRR)

Usage

```
zQRR(Q.true, Q.sug)
```

Arguments

Q.true	The true Q-matrix.
Q.sug	The Q-matrix that has been validated.

Details

The Q-matrix recovery rate (QRR) provides information on overall performance, and is defined as:

$$QRR = \frac{\sum_{i=1}^{I} \sum_{k=1}^{K} I(q_{ik}^{t} = q_{ik}^{s})}{IK}$$

where q_{ik}^{t} denotes the kth attribute of item i in the true Q-matrix ($Q.true$), q_{ik}^{s} denotes kth attribute of item i in the suggested Q-matrix($Q.sug$), and $I(\cdot)$ is the indicator function.

Value

A numeric (QRR index).

Examples

```
library(Qval)

set.seed(123)

Q1 <- sim.Q(5, 30)
Q2 <- sim.MQ(Q1, 0.1)
QRR <- zQRR(Q1, Q2)
print(QRR)
```

zTNR *Calculate True-Negative Rate (TNR)*

Description

Calculate True-Negative Rate (TNR)

Usage

```
zTNR(Q.true, Q.orig, Q.sug)
```

Arguments

Q.true	The true Q-matrix.
Q.orig	The Q-matrix need to be validated.
Q.sug	The Q-matrix that has been validated.

Details

TNR is defined as the proportion of correct elements which are correctly retained:

$$TNR = \frac{\sum_{i=1}^{I}\sum_{k=1}^{K} I(q_{ik}^{t} = q_{ik}^{s} | q_{ik}^{t} \neq q_{ik}^{o})}{\sum_{i=1}^{I}\sum_{k=1}^{K} I(q_{ik}^{t} \neq q_{ik}^{o})}$$

where q_{ik}^{t} denotes the kth attribute of item i in the true Q-matrix (Q.true), q_{ik}^{o} denotes kth attribute of item i in the original Q-matrix(Q.orig), q_{ik}^{s} denotes kth attribute of item i in the suggested Q-matrix(Q.sug), and $I(\cdot)$ is the indicator function.

Value

A numeric (TNR index).

Examples

```
library(Qval)

set.seed(123)

Q1 <- sim.Q(5, 30)
Q2 <- sim.MQ(Q1, 0.1)
Q3 <- sim.MQ(Q1, 0.05)
TNR <- zTNR(Q1, Q2, Q3)

print(TNR)
```

zTPR *Calculate True-Positive Rate (TPR)*

Description

Calculate True-Positive Rate (TPR)

Usage

```
zTPR(Q.true, Q.orig, Q.sug)
```

Arguments

Q.true	The true Q-matrix.
Q.orig	The Q-matrix need to be validated.
Q.sug	The Q-matrix that has been validated.

Details

TPR is defined as the proportion of correct elements which are correctly retained:

$$TPR = \frac{\sum_{i=1}^{I}\sum_{k=1}^{K} I(q_{ik}^{t} = q_{ik}^{s} | q_{ik}^{t} = q_{ik}^{o})}{\sum_{i=1}^{I}\sum_{k=1}^{K} I(q_{ik}^{t} = q_{ik}^{o})}$$

where q_{ik}^{t} denotes the kth attribute of item i in the true Q-matrix (Q.true), q_{ik}^{o} denotes kth attribute of item i in the original Q-matrix(Q.orig), q_{ik}^{s} denotes kth attribute of item i in the suggested Q-matrix(Q.sug), and $I(\cdot)$ is the indicator function.

Value

A numeric (TPR index).

Examples

```
library(Qval)

set.seed(123)

Q1 <- sim.Q(5, 30)
Q2 <- sim.MQ(Q1, 0.1)
Q3 <- sim.MQ(Q1, 0.05)
TPR <- zTPR(Q1, Q2, Q3)

print(TPR)
```

zUSR *Calculate Under-Specification Rate (USR)*

Description

Calculate Under-Specification Rate (USR)

Usage

```
zUSR(Q.true, Q.sug)
```

Arguments

Q.true	The true Q-matrix.
Q.sug	The Q-matrix that has been validated.

Details

The USR is defined as:

$$USR = \frac{\sum_{i=1}^{I} \sum_{k=1}^{K} I(q_{ik}^{t} > q_{ik}^{s})}{IK}$$

where q_{ik}^{t} denotes the kth attribute of item i in the true Q-matrix (Q.true), q_{ik}^{s} denotes kth attribute of item i in the suggested Q-matrix(Q.sug), and $I(\cdot)$ is the indicator function.

Value

A numeric (USR index).

Examples

```
library(Qval)

set.seed(123)

Q1 <- sim.Q(5, 30)
Q2 <- sim.MQ(Q1, 0.1)
USR <- zUSR(Q1, Q2)
print(USR)
```

zVRR *Calculate Vector Recovery Ratio (VRR)*

Description

Calculate Vector Recovery Ratio (VRR)

Usage

```
zVRR(Q.true, Q.sug)
```

Arguments

Q.true	The true Q-matrix.
Q.sug	The Q-matrix that has been validated.

Details

The VRR shows the ability of the validation method to recover q-vectors, and is determined by

$$VRR = \frac{\sum_{i=1}^{I} I(\mathbf{q}_i^t = \mathbf{q}_i^s)}{I}$$

where $\mathbf{q}_i^t$ denotes the $\mathbf{q}$-vector of item i in the true Q-matrix (Q.true), $\mathbf{q}_i^s$ denotes the $\mathbf{q}$-vector of item i in the suggested Q-matrix(Q.sug), and $I(\cdot)$ is the indicator function.

Value

A numeric (VRR index).

Examples

```
library(Qval)

set.seed(123)

Q1 <- sim.Q(5, 30)
Q2 <- sim.MQ(Q1, 0.1)
VRR <- zVRR(Q1, Q2)
print(VRR)
```

Index