

Package: bigIRT (via r-universe)

October 12, 2026

Title Item Response Theory for Large Sparse Data

Version 0.2.0

Description Fits binary item response theory (IRT) models for large and sparse data in long format, by marginal maximum likelihood with a Laplace approximation to the integral over person abilities, or by the joint posterior mode over items and persons. Supports 1PL to 4PL models, multidimensional and confirmatory loading structures with an estimated latent correlation, covariate effects on ability and on each item parameter, empirical-Bayes prior adaptation, and post-estimation tools for information, reliability, item fit, held-out prediction and simulation.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 3.5.0), data.table

Imports Matrix, methods, mize, Rcpp (>= 0.12.0), RcppParallel (>= 5.0.1), splines

LinkingTo Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0), RcppParallel (>= 5.0.1)

Suggests broom, ggplot2, loo, mirt, testthat, knitr, rmarkdown, sandwich, strucchange

VignetteBuilder knitr

SystemRequirements GNU make

URL <https://github.com/cdriveraus/bigIRT>

BugReports <https://github.com/cdriveraus/bigIRT/issues>

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make

Repository <https://cdriveraus.r-universe.dev>

Date/Publication 2026-10-10 09:55:55 UTC

RemoteUrl <https://github.com/cdriveraus/bigIRT>

RemoteRef HEAD

RemoteSha 7eef27fe82a37aeb0ae532f254c48eb7bdd433a5

Contents

as.mirt	3
augmentIRT	3
checkConvergence	4
checkMirtBridge	5
compareIRTmodels	5
dropPerfectScores	6
empiricalICC	7
estfun.bigIRT_scores	8
extractMIRTpars	10
fitIRT	10
glanceIRT	23
heldoutLogLik	24
heldoutMetrics	24
improveModel	25
IRTcurve	27
IRTic	28
itemFit	28
itemInformation	29
logLik.bigIRT_fit	30
looIRT	31
modificationTests	32
normaliseIRT	33
normaliseMIRT	34
personCovarianceMatrices	36
personCovarianceMatrices_R	38
plotLaplaceDiagnostics	39
reliability	40
reportIRT	40
rowLoadingsToSparse	41
scoreContributions	42
scoreTest	43
simIRT	47
testInformation	50
tidyIRT	50
wleIRT	51

Index	53
--------------	-----------

as.mirt

*Coerce a fit to a fixed-parameter mirt object***Description**

Opens mirt's diagnostic suite – `itemfit()`, `personfit()`, `M2()`, `residuals(type = "Q3")`, `itemplot()`, `DIF()` – none of which is worth reimplementing, because all of them need the wide person-by-item matrix and that, not the arithmetic, is the binding constraint. Data too large to hold as one cannot cross, and is refused rather than attempted.

Usage

```
as.mirt(fit, max_cells = 5e+07, verbose = FALSE)
```

Arguments

<code>fit</code>	A fitted object from <code>fitIRT()</code> .
<code>max_cells</code>	Numeric. Refuse a wide matrix larger than this.
<code>verbose</code>	Logical. Passed to <code>mirt::mirt()</code> .

Value

A `SingleGroupClass` object with every parameter fixed at this fit's values.

See Also

`checkMirtBridge()`, which verifies the translation.

augmentIRT

*Per-response fitted values and residuals***Description**

Returned in the order the data were supplied, so the result can be bound back onto the caller's own data frame. Registered against `broom::augment()` when `broom` is installed.

Usage

```
augmentIRT(x, ...)
```

Arguments

<code>x</code>	A fitted object from <code>fitIRT()</code> .
<code>...</code>	Unused.

Value

A data frame of row index, observed response, fitted probability, training flag, residual and standardised residual.

checkConvergence	<i>Convergence, resolved by parameter block</i>
------------------	---

Description

A single scaled gradient norm cannot say which block failed to settle, it hides one stuck parameter among thousands, and a small block whose gradient aggregates over many units can dominate it. This reports per-block norms and shares, the worst parameter measured in its own standard errors, and the Newton decrement, which estimates how much objective is still available.

Usage

```
checkConvergence(fit, cores = 1L, se = TRUE)
```

Arguments

fit	A fitted object from <code>fitIRT()</code> .
cores	Integer. Cores for the information calculation.
se	Logical. Compute the item information needed for the Newton decrement and the worst-parameter measure.

Value

An object of class `bigIRT_convergence`, carrying `blocks`, `newton_decrement`, `objective_remaining`, `worst_parameter_se_units` and the fit's certificate.

Note

The decrement uses the block-diagonal item curvature, the same model the Newton polish steps in, so it is self-certifying. Compare decrements within one algorithm, and compare algorithms by the objective. The certificate a Laplace fit carries (`fitIRT(laplaceCertify = )`) is computed on the exact Hessian of the fitted objective instead; it is returned as `certificate`, and it is the verdict converged follows.

checkMirtBridge	<i>Verify the mirt coercion reproduces this fit's curves</i>
-----------------	--

Description

Compares mirt's own trace lines against bigIRT's response function on a grid of abilities. This tests the parameter translation directly, without marginalisation, so a coercion error is separated from the difference between Laplace and quadrature. A coercion that is quietly wrong produces diagnostics that look entirely reasonable, which is why this exists.

Usage

```
checkMirtBridge(
  fit,
  mirt_fit = NULL,
  theta = seq(-3, 3, 0.5),
  tolerance = 1e-06,
  max_items = 25L
)
```

Arguments

fit	A fitted object from <code>fitIRT()</code> .
mirt_fit	Optional object from <code>as.mirt()</code> ; built if absent.
theta	Numeric vector of ability points.
tolerance	Numeric. Largest acceptable probability difference.
max_items	Integer. Items to check.

Value

An object of class `bigIRT_mirtcheck`.

compareIRTmodels	<i>Compare IRT Model Parameterizations</i>
------------------	--

Description

Coerces simulated truth, bigIRT fits, and mirt fits onto a common raw and normalized parameter representation for direct comparison.

Usage

```
compareIRTmodels(
  models,
  score_method = "EAP",
  normaliseScale = 1,
  normaliseMean = 0,
  curveSampleN = 50L
)
```

Arguments

<code>models</code>	A named or unnamed list containing up to one <code>simIRT</code> object and any number of fitted <code>bigIRT</code> and/or <code>mirt</code> models.
<code>score_method</code>	Character string passed to <code>mirt::fscores()</code> when extracting abilities from <code>mirt</code> fits.
<code>normaliseScale</code>	Numeric scalar passed to <code>normaliseMIRT()</code> .
<code>normaliseMean</code>	Numeric scalar or vector passed to <code>normaliseMIRT()</code> .
<code>curveSampleN</code>	Integer. Number of multivariate-normal theta draws used to compute item-response-curve RMSE against the reference model for each item and form. Default is 50.

Details

Which form to read: when every model identifies its metric the way the reference does – the same fixed latent distribution, a confirmatory loading pattern – the "raw" form is already on the reference's metric and is the one to compare. The "normalized" form standardises each model by the spread of its own ability estimates, which for a fitted model are point estimates shrunk toward the prior mean: it flattens the fitted loadings by roughly the square root of the reliability (about a fifth at ten responses per person) while the reference, standardised by its true abilities, is left alone, and so favours models with steeper loadings. Use it only when the metrics genuinely differ, and treat its curve RMSE accordingly.

Value

A list with elements `itempars`, `personpars`, `loading_matrix`, `ability_corr_matrix`, `raw`, `normalized`, and `reference_model`. Each table is in long format with a source column, a source-specific value column, and a `referenceVal` column holding the aligned value from the reference model.

<code>dropPerfectScores</code>	<i>Drop subjects and items with all perfect scores</i>
--------------------------------	--

Description

This function drops variables/items and subjects that have all perfect scores (either all 0's or all 1's) in a data table.

Usage

```
dropPerfectScores(  
  dat,  
  scoreref. = "score",  
  itemref. = "Item",  
  idref. = "id",  
  tol. = 0.001,  
  trainref. = NULL  
)
```

Arguments

dat	The input data table
scoreref.	The column name of the score variable in dat
itemref.	The column name of the item variable in dat
idref.	The column name of the id variable in dat
tol.	Tolerance level for checking perfect scores – .01 would drop subjects with less than 1% correct or incorrect
trainref.	Optional name of a 0/1 column marking training rows. When given, whether a person or item is perfect is decided from its training rows only, and all its rows (held-out ones too) are dropped; a person or item with no training rows is kept. fitIRT passes its trainingRows this way, so held-out responses never decide who is fitted.

Value

The input data table (dat) without variables/items and subjects with all perfect scores.

Examples

```
dat <- data.table(id=c(1,1,1,2,2,2,3,3,3), Item=c('I1','I2','I3','I1','I2','I3','I1','I2','I3'),  
  score=c(1,0,1,0,0,0,0,0,1,1))  
print(dropPerfectScores(dat))
```

empiricalICC	<i>Observed against model-implied response curves</i>
--------------	---

Description

People are binned by fitted ability and the observed proportion correct in each bin is compared with the model’s prediction, which says how an item misfits rather than only that it does.

Usage

```
empiricalICC(
  fit,
  items = NULL,
  bins = 10L,
  scale_index = 1L,
  training_only = TRUE
)
```

Arguments

<code>fit</code>	A fitted object from <code>fitIRT()</code> .
<code>items</code>	Optional item index to restrict to.
<code>bins</code>	Integer. Number of ability bins.
<code>scale_index</code>	Integer. Latent dimension to bin on.
<code>training_only</code>	Logical. Use training rows only.

Value

A data frame of item, bin, count, mean ability, observed and expected proportion, and a Wilson interval.

Note

Bins are formed on estimated ability, which is shrunk and measured with error, so people sort partly by noise and the observed curve flattens against the model. On correctly specified data the model falls inside the nominal 95 per cent interval for about 80 per cent of bins. Read a mild symmetric shortfall as that artefact, and a one-sided or localised departure as misfit.

`estfun.bigIRT_scores` *Score contributions as an estimating-function matrix*

Description

Arranges the score contributions of `scoreContributions()` as the matrix `sandwich::estfun()` returns: one row per person (the person's marginal score, summed over their responses) or per response, one column per free item parameter. Tools built on `estfun`, such as the generalised fluctuation tests of `strucchange::gefp()` and `strucchange::sctest()`, then run on a `bigIRT` fit as they do on `mirt` fits.

Usage

```
estfun.bigIRT_scores(
  x,
  items = NULL,
  parameters = c("B", "A", "C", "D"),
  level = c("person", "row"),
  ...
)
```

Arguments

<code>x</code>	A <code>bigIRT_scores</code> object.
<code>items</code>	Items to include (default all). Each free parameter is a column, so for large banks choose items.
<code>parameters</code>	Parameter types to include: any of "B", "A" (all loadings), "C", "D".
<code>level</code>	"person" (default) or "row".
<code>...</code>	Unused.

Details

Persons are independent and are the units such tools assume; responses within a person are not, so `level = "row"` is for bookkeeping (a covariate given per response), not for tests that treat rows as independent – `scoreTest()` handles those. Those tools decorrelate with the outer product of these scores; `scoreTest()` uses each response's expected information instead, which on thin items is better calibrated.

Value

A matrix with rows in the order of `x$persons` (or `x$rows`) and columns named `item:parameter`.

Examples

```
## Not run:
sc <- scoreContributions(fit)
z <- covariate_per_person[as.integer(sc$person_labels)]
gp <- strucchange::gefp(sc, fit = NULL, order.by = z,
  scores = function(x) sandwich::estfun(x, items = "item7"))
strucchange::sctest(gp, functional = "DM")

## End(Not run)
```

extractMIRTpars	<i>Extract Comparable MIRT Parameters</i>
-----------------	---

Description

Extracts a common set of multidimensional IRT parameters from either a bigIRT fit or a `mirt` fit.

Usage

```
extractMIRTpars(object, score_method = "EAP")
```

Arguments

<code>object</code>	A fitted bigIRT object or a mirt SingleGroupClass object.
<code>score_method</code>	Character string passed to <code>mirt::fscores()</code> when extracting abilities from a mirt fit.

Value

A list with components `source`, `A`, `B`, `C`, `D`, `Ability`, `corr`, and `latent_mean` and `latent_sd`: the mean and standard deviation of the latent variable on the solution's own metric, which is what a normalisation should use. `Ability` holds point estimates, whose spread understates the latent scale. For a bigIRT fit they come from the posterior by the law of total variance (variance of the posterior modes plus the mean posterior variance), for a mirt fit from its estimated latent distribution; they are NULL for a simulation, whose `Ability` is the latent variable itself, and for a fit that kept no posterior precision.

fitIRT	<i>Fit a binary Item Response Theory (IRT) model</i>
--------	--

Description

Fits a 1PL to 4PL model to long-format binary response data, in one or more dimensions, with optional covariates on ability and on every item parameter. Two estimators are available through `marginalApprox`, and they take the same arguments and return the same kind of object:

Usage

```
fitIRT(
  dat,
  score = "score",
  id = "id",
  item = "Item",
  scale = "Scale",
  pl = 1,
```

```

marginalApprox = c("laplace", "map", "none", "laplace_fast", "laplace_direct"),
personDat = NA,
personPreds = character(),
itemDat = NA,
loadings = NA,
loadingsFixed = NA,
signedLoadings = NULL,
AitemPreds = character(),
BitemPreds = character(),
CitemPreds = character(),
DitemPreds = character(),
itemSpecificBetas = FALSE,
betaScale = 10,
priors = TRUE,
invspAMeandat = 0.542,
invspASD = 1,
BMeandat = 0,
BSD = 2.5,
logitCMeandat = 0,
logitCSD = 1,
logitDMeandat = 0,
logitDSD = 1,
AbilityMeandat = array(0, dim = c(length(unique(dat[[scale]])))),
AbilitySD = array(1, dim = c(length(unique(dat[[scale]])))),
AbilityCorr = diag(1, c(length(unique(dat[[scale]])))),
AMeanSD = 1,
BMeanSD = BSD,
logitCMeanSD = logitCSD,
logitDMeanSD = logitDSD,
AbilityMeanSD = array(1, dim = c(length(unique(dat[[scale]])))),
estMeans = c("A", "B", "C", "D"),
estimateAbilityCorr = FALSE,
laplaceCorrParam = c("stan_corsqrt", "normalized_chol"),
ebayes = FALSE,
ebayesMethod = c("laplace", "moment"),
ebayesIter = 5L,
ebayesCoarse = 1000,
ebayesMinSD = 0.05,
ebayesmultiplier = 2,
laplaceRefine = c("auto", "agq", "laplace2", "none"),
agqNodes = NULL,
agqPoints = NULL,
laplaceOptimizer = c("fisher", "mize"),
laplaceOuterIter = 500,
laplaceTol = 1e-04,
laplaceGradTol = 1e-04,
laplaceTolScale = c("relative", "per_obs"),
laplacePolish = NA,

```

```

    laplacePersonTol = 1e-06,
    laplaceCertify = TRUE,
    laplaceJitter = 1e-06,
    laplaceKeepCovariance = FALSE,
    laplaceDiagnostics = FALSE,
    laplacePlot = FALSE,
    laplacePlotEvery = 1L,
    normalise = FALSE,
    normaliseScale = 1,
    normaliseMean = 0,
    dropPerfectScores = TRUE,
    trainingRows = 1:nrow(dat),
    cores = 6,
    verbose = 0L,
    keepInternals = FALSE,
    ...
  )

```

Arguments

<code>dat</code>	A data frame containing the data to be analyzed.
<code>score</code>	Character. The name of the column in <code>dat</code> representing the response. Default is 'score'.
<code>id</code>	Character. The name of the column in <code>dat</code> representing the individual. Default is 'id'.
<code>item</code>	Character. The name of the column in <code>dat</code> representing the item. Default is 'Item'.
<code>scale</code>	Character. The name of the column in <code>dat</code> representing the scale of each item. Default is 'Scale'.
<code>pl</code>	Integer. The number of parameters for the logistic model (1PL, 2PL, 3PL, or 4PL). Default is 1.
<code>marginalApprox</code>	Character. The estimator. "laplace" (default): marginal maximum likelihood, the abilities integrated out. The optimiser works over the item-side parameters and re-solves every person mode inside each objective evaluation, differentiating through them, so the gradient carries the mode-adjoint term rather than treating the modes as fixed. It accepts item predictors, person predictors, estimated ability means and estimated latent correlations, and its gradient agrees with central finite differences in every one of those blocks. The Laplace curvature at the person mode is the <i>observed</i> Hessian, the actual second derivative of the log posterior. Fisher information was used until September 2026 and was selectable for a while after; it is neither now. Two reasons. Laplace approximates one integrand, so it needs that integrand's curvature rather than an average over responses the person did not give. And the mode adjoint applies the implicit function theorem, which needs the inverse of the actual Hessian: with Fisher in that role the item gradient missed the true total derivative by 1e-3 to 4e-2 for the 3PL and 4PL, at one, two and three dimensions. The

2PL is exact either way, because the logit link is canonical and the two curvatures are then the same number – which is why it went unnoticed for so long. Fisher information is still used for the Newton step, where its guaranteed positive definiteness is what makes the mode solve robust. Where the data’s own contribution to the curvature is not positive semidefinite, it is clipped to the nearest matrix that is, rather than swapped for Fisher: swapping is discontinuous in the item parameters and a line search cannot cope with it. A consequence worth knowing is that the reported person posterior SD can never exceed the ability prior SD.

"map": the joint posterior mode over item parameters and abilities. The persons are profiled out – each evaluation solves every person’s mode and differentiates the joint log posterior at it, which needs no adjoint because the abilities are at their maximum (`src/map_backend.cpp`) – and the result is optimised, certified and reported exactly as a Laplace fit is. It takes every argument the Laplace estimator takes except `estimateAbilityCorr`, and ignores `laplaceRefine` except to choose how its marginal likelihood is evaluated for `logLik()`. Two properties to know before using it. Its latent scale comes from the priors, not the data: the likelihood is unchanged by scaling every loading up and every ability down together, and the joint mode trades the persons’ ability prior against the items’ loading prior, so loadings come out inflated and abilities shrunk, more so the more persons there are per item (on simulated 2PL data with loadings near 1.2, joint-mode loadings averaged 2.3 to 3.8). Compare it with other fits only after linking or normalising (`normalise`, [normaliseIRT](#)). And it cannot estimate the latent correlation; see `estimateAbilityCorr`. This replaces, in 0.2.0, a Stan implementation of the same estimator that optimised every ability and item parameter jointly by L-BFGS; on the same data the two agree to the optimiser’s tolerance, and the joint posterior at this implementation’s solution was the higher.

Predictor columns are response-row aligned and can vary within a person or item, under either estimator; the likelihood retains their exact row values while reported person and item summaries use within-entity predictor means. "none" is accepted as the former name of "map", and "laplace_fast" and "laplace_direct" as former names of "laplace".

<code>personDat</code>	Data frame. Any fixed ability data for persons. Default is NA.
<code>personPreds</code>	Character vector. Names of predictors for person parameters found in data. Default is an empty character vector.
<code>itemDat</code>	Data frame. Any fixed item data. Default is NA.
<code>loadings</code>	Optional numeric matrix (<code>Nitems</code> x <code>Nscales</code>) defining item-by-scale discrimination structure. Each cell can be: • a numeric value (fixed loading), • NA (free loading, estimated), • or omitted by leaving the whole argument as NA, which uses the legacy one-hot bridge from scale.

Row names are matched to item ids and column names are matched to scale ids when provided.

loadingsFixed	Optional logical matrix (Nitems x Nscales) that forces fixed/free status for each loading entry. TRUE means fixed, FALSE means free, NA defers to loadings values. If omitted, fixed/free status is inferred from whether loadings entries are NA.
signedLoadings	Which free loadings may take either sign. Every free loading is otherwise the softplus $\log(1 + e^x)$ of an unconstrained parameter, so it is positive. That positivity is what fixes the sign of each latent dimension, and it is kept for primary loadings; but it makes a negative cross-loading impossible, so an item that truly loads negatively on a second dimension has its cross-loading pinned near zero instead. A signed loading takes the identity link: the loading is the parameter itself. • NULL or FALSE (default): none; exactly the behaviour without the argument. • "cross": every free loading off the item's own scale (its scale column) is signed; primary loadings stay positive. • a logical Nitems x Nscales matrix, aligned by row and column names as loadings is: TRUE marks a signed loading. Marking a fixed loading is an error. A signed loading's prior is normal with mean zero and SD invspASD, on the loading itself. Its mean is fixed: the estimated discrimination mean (estMeans 'A') describes the softplus pre-images of the positive loadings and applies to those alone, and empirical Bayes leaves signed loadings out of its discrimination update. It starts at zero. Item covariates on discrimination (AitemPreds) add to the parameter under either link. Requires p1 >= 2. Signs do not identify a rotation: with cross-loadings free in both directions the dimensions can rotate into each other unless loadings fixes enough of them, at least $K(K - 1)/2$ zeros placed so that every pair of dimensions is pinned. tidyIRT() reports free cross-loadings as terms A_<scale> on fits with signed loadings.
AitemPreds	Character vector. Names of predictors for item discrimination parameters. Default is an empty character vector.
BitemPreds	Character vector. Names of predictors for item difficulty parameters. Default is an empty character vector.
CitemPreds	Character vector. Names of predictors for item guessing parameters. Default is an empty character vector.
DitemPreds	Character vector. Names of predictors for item upper asymptote (slipping) parameters. Default is an empty character vector.
itemSpecificBetas	Logical. Whether to allow item-specific betas for covariate effects, or simply estimate one effect per covariate. Default is FALSE.
betaScale	Numeric. Scale of the prior for beta parameters. Default is 10.
priors	Logical. Whether to use prior distributions. Default is TRUE.
invspAMEandat	Numeric. Mean for the prior distribution of the raw discrimination parameters, which subsequently have a 'softplus' $\log(1+\exp(x))$ applied. Default is 0.542, giving a mean for A pars of ~ 1.

invspASD	Numeric. Standard deviation for the prior distribution of the raw discrimination parameters. Default is 1.
BMeandat	Numeric. Mean for the prior distribution of the item difficulty parameters. Default is 0.
BSD	Numeric. Standard deviation for the prior distribution of the item difficulty parameters. Default is 2.5, on the scale of AbilitySD. The previous default of 10 was effectively flat: on a standardised ability scale it puts about 69 per cent of its mass beyond $ B > 4$, where an item is answered by almost everybody or almost nobody. Difficulty is well identified in a 2PL and barely notices, but under a 3PL or 4PL it absorbs the error in a weakly identified asymptote and runs away – measured difficulty RMSE above 3 on sparse 4PL cells, against 0.4 with this default. Widen it for vertical scales, whose difficulties genuinely span more than this.
logitCMeandat	Numeric. Mean for the prior distribution of the item guessing parameters (on logit scale). Default is 0.
logitCSD	Numeric. Standard deviation for the prior distribution of the item guessing parameters (on logit scale). Default is 1. The previous default of 5 carried a prior precision of 0.04, which is negligible against any data, so per-item guessing was effectively unconstrained; 1 gives precision comparable to what a handful of responses provide. Note that guessing is generally not recoverable per item at realistic test lengths – fitted values correlate with the truth at around .07 on 20 responses an item – so this prior mostly decides how firmly items pool towards a common asymptote.
logitDMeandat	Numeric. Mean for the prior distribution of the item upper asymptote parameters (on logit scale). Default is 0.
logitDSD	Numeric. Standard deviation for the prior distribution of the item upper asymptote parameters (on logit scale). Default is 1, for the reasons given under logitCSD; the upper asymptote is usually pinned by even fewer responses than the lower one.
AbilityMeandat	Numeric array. Mean for the prior distribution of the ability parameters. Default is 0 for each scale.
AbilitySD	Numeric array. Standard deviation for the prior distribution of the ability parameters. Default is 1 for each scale.
AbilityCorr	Matrix. Correlation matrix for the ability parameters. Default is an identity matrix.
AMeanSD	Numeric. Standard deviation for the prior distribution of the discrimination parameters. Default is 1.
BMeanSD	Numeric. Standard deviation for the prior distribution of the difficulty parameters. Default is BSD.
logitCMeanSD	Numeric. Standard deviation for the prior distribution of the guessing parameters (on logit scale). Default is logitCSD.
logitDMeanSD	Numeric. Standard deviation for the prior distribution of the upper asymptote parameters (on logit scale). Default is logitDSD.
AbilityMeanSD	Numeric array. Standard deviation for the prior distribution of the ability parameters. Default is 1 for each scale.

estMeans	Character vector. Which means to estimate, from 'ability', 'A', 'B', 'C', 'D'. Default is <code>c('A', 'B', 'C', 'D')</code>: the item means are estimated and the ability mean is held at <code>AbilityMeandata</code>, which is 0 unless supplied. The ability mean and the difficulty mean cannot both be estimated. Adding a constant to every ability and the same constant to every difficulty leaves every response probability unchanged, so the location of the scale has to be pinned by one or the other. The default pins it through the ability prior; passing 'ability' here estimates the ability mean instead and requires dropping 'B'. This matters when whole persons are held out of <code>trainingRows</code>: such a person has no likelihood, so their ability is the prior mean, which under the default is 0 plus whatever their <code>personPreds</code> imply.
estimateAbilityCorr	Logical. If TRUE, estimate the latent ability correlation matrix while keeping <code>AbilitySD</code> fixed, so that only the correlation is free (the latent scale is not identified separately from the loadings). The Laplace estimator optimises it as part of the parameter vector. Under <code>marginalApprox = "map"</code> it is an error: as a parameter of the joint posterior the correlation runs to the boundary (the prior's normalising constant grows without bound as a correlation goes to one, while the likelihood need not fall), and set by EM from the person posteriors it was measured biased towards zero (.39 to .50 for a true .6), because the joint mode's latent scale is not the prior's. Ignored for unidimensional fits. It forces <code>laplaceKeepCovariance</code>, because the person posterior covariances are required: modes are shrunk towards zero by the prior, so their cross-product alone understates the correlation. The result is returned in <code>fit\$abilityPrior\$corr</code>, with <code>fit\$abilityPrior\$estimated</code> recording whether it was estimated or held fixed. This prior-level correlation is the quantity to report: the correlation among fitted ability point estimates is biased, downwards under an independent prior and upwards under a correlated one. Default is FALSE.
laplaceCorrParam	Character. Parameterisation used for the latent correlation when <code>estimateAbilityCorr = TRUE</code>.
ebayes	Logical. Whether to use empirical Bayes estimation: the priors are adapted from a first-pass estimate. Default FALSE since 0.1.9 (TRUE before). On simulated 3PL and 4PL data, empirical-Bayes priors narrowed toward the first pass left item intervals covering .004 to .84 of the time against a nominal .95, where the fixed default priors were conservative.
ebayesMethod	Character. How prior hyperparameters when <code>ebayes = TRUE</code>. "laplace" (default) approximates the integral over the item parameters and maximises the resulting profile marginal, which is bounded and needs no floor or multiplier. "moment" keeps the earlier variance-components rule, which is faster but relies on a floor.
ebayesIter	Integer. Maximum empirical-Bayes rounds. Each round refits, updates the prior hyperparameters from the fit by the EM M-step for a normal-normal hierarchy, and stops early once every hyperparameter moves by less than a thousandth. Ignored when <code>ebayes = FALSE</code>. Default is 5.
ebayesCoarse	Numeric. Tolerance multiplier for the intermediate empirical-Bayes refits. Those rounds exist only to produce estimates to update the hyperparameters from, so

they are run at `laplaceTol * ebayesCoarse`; the final fit always uses `laplaceTol`. Default is 1000, so that with the default `laplaceTol` the rounds stop at 0.1 nats as before (the defaults were 100 and 1e-3 until September 2026).

<code>ebayesMinSD</code>	Numeric. Floor on a prior SD estimated by <code>ebayesMethod = "moment"</code> . That rule subtracts a sampling-noise term and reaches zero whenever the data cannot support item-level variation, so the floor decides how hard those blocks pool. It does not apply to "laplace", which is bounded by its own objective. Default is 0.05.
<code>ebayesmultiplier</code>	Numeric. Multiplier on the prior SDs estimated by <code>ebayesMethod = "moment"</code> . Default is 2.
<code>laplaceRefine</code>	Character. Laplace estimator only (for "map" it picks only how <code>logLik()</code> 's marginal likelihood is evaluated). Whether to finish a "laplace" fit on the marginal likelihood with each person's integral done by adaptive Gauss-Hermite quadrature, centred at the Laplace mode, instead of by the Laplace approximation. The refinement starts from the Laplace optimum and uses the same optimiser, so it is a short continuation, not a second fit. "auto" (default): with free asymptotes (3PL, 4PL), by quadrature at any dimension; without, not at all in one dimension, by quadrature in two and three, and by "laplace2" beyond three, where Laplace put 2PL loadings 0.12 to 0.22 standard errors low on simulated data and quadrature costs about four times as much (earlier versions used "laplace2" from two dimensions on; see below for why that changed). "agq" always refines by quadrature: a product Gauss-Hermite rule up to three dimensions, and beyond that an expansion of each person's log integral in product rules on every pair (and, coarsely, every triple) of dimensions, which on 3PL data at nine dimensions, where Laplace was up to seven standard errors off, came within 0.5 of a 16,384-point reference at a twelfth of its cost, and on 4PL data at five dimensions (Laplace 13 off) within 2 (quasi-Monte Carlo points instead, with <code>agqPoints</code> , need many thousands to be reliable: a 256-point refinement at nine dimensions was measured running away). "laplace2" refines by the second-order Laplace approximation, each person's Laplace value plus its leading correction from the third and fourth derivatives, which costs the same at any dimension; at nine dimensions with cross loadings it cut the per-person error of the log marginal from 0.095 to 0.004 nats, against 0.012 for 256 quasi-Monte Carlo points. "none" never refines. Why: Laplace is biased when asymptotes are free. Measured on simulated 3PL data, the exact-marginal optimum sat about half a standard error above the Laplace one in every discrimination (loadings about 5 per cent larger), and on 4PL data nearly one standard error (8 to 13 per cent); the bias is fixed per person while standard errors shrink with N, so intervals under-cover more as data grow. Laplace can also hold a fit at an optimum the exact marginal does not have. For the one-dimensional 2PL the shift was about 0.03 standard errors, which is why "auto" leaves it alone – under the item priors. Without them (<code>priors = FALSE</code>) the 2PL Laplace marginals reward a near-step curve for a weakly identified item: on sparse simulated data with 75 training responses per item, 75 per cent of one-dimensional Laplace fits had a discrimination above 10, against 10 per cent of the same fits finished by quadrature and 5 per cent for an EM quadrature fit (<code>paper/Submission2</code> , <code>unregularised_marginal_check.R</code>). Set "agq" when the item priors are off. Without

them an item can also run off along its own ray, its loading and intercept growing together towards a step, where the gradient along the ray vanishes. So the refinement judges each stretch of steps taken on held quadrature nodes against the objective with the nodes re-centred, and undoes one that lost more than re-centring alone moves it (held nodes are a discrete ability distribution, which a step between two of them fits better the steeper it is); and it restarts from run-off items pulled back along their rays, keeping whichever end is best on its own objective. `laplaceStatus$refine` records both (`held_nodes_lost`, `pulled_back`). The second-order finish also falls short, with the priors on, when each person has only a few responses per dimension. On sparse three-dimensional 2PL data with about eight fitted responses per person, its primary loadings had an RMSE of .283 against .235 for the same fits finished by quadrature, and sat higher (mean error +.045 against +.020), in every one of 20 replications; and with 1,350 persons, 2 to 4 per cent of its fits were certified at a spurious optimum with one item at a discrimination of 4 to 10, where quadrature put it at 1.6 and an EM fit below 2 (paper/Submission2, Study 2). That is why "auto" now finishes the 2PL by quadrature up to three dimensions, at four to five times the former cost there. At nine dimensions the same data showed the smaller, systematic part of the problem (curves recovered slightly less closely than by quadrature, no spurious optima in ten fits), and quadrature there costs about four times as much; set "agq" for sparse high-dimensional fits when accuracy matters more than speed. The reported `logLik` is then the quadrature marginal; `laplaceStatus` records both values and what the refinement did.

<code>agqNodes</code>	Integer. Gauss-Hermite nodes per dimension for <code>laplaceRefine</code> . Up to three dimensions the product rule costs agqNodes^K likelihood passes per person, so the default falls with dimension: 9, 7 and 5 for one, two and three dimensions. Beyond three, the pairwise expansion costs about $K^2 \text{agqNodes}^2 / 2$ nodes per person, default 7, plus a three-way correction at 3 nodes per dimension (<code>options(bigIRT.agq_nodes3 = 0)</code> drops it), about 2,200 nodes in all at nine dimensions.
<code>agqPoints</code>	Integer. If given, beyond three dimensions the refinement uses this many quasi-Monte Carlo points per person instead of the pairwise expansion, in antithetic pairs, as importance sampling through the Laplace Gaussian. Held fixed during the fit they can be exploited unless there are many thousands. Default NULL.
<code>laplaceOptimizer</code>	Character. Optimiser, under either estimator. "fisher" (default) is L-BFGS whose initial inverse Hessian is shaped by the Fisher information of the marginal likelihood, including the coupling between items that runs through the persons, and which stops when the predicted gain in log-likelihood falls below <code>laplaceTol</code> nats. "mize" is the previous path: mize's L-BFGS stopping on the change in objective, followed by the block-Newton polish. Measured on simulated 2PL to 4PL designs, the Fisher path needed about a third of the objective evaluations and half the time, and ended closer to the optimum. It builds the information as a sparse matrix, and falls back to "mize" when that would exceed about $5e7$ nonzeros.
<code>laplaceOuterIter</code>	Integer. Maximum optimiser iterations, under either estimator. This is a fallback limit rather than the primary convergence criterion. Default is 500.

laplaceTol	Numeric. Optimiser tolerance, under either estimator. Under <code>laplaceOptimizer = "fisher"</code> it is the predicted remaining gain in log-likelihood, in nats, at which the optimiser stops; under <code>"mize"</code> it is <code>mize</code> 's absolute tolerance on the change in objective and on the gradient norm. Default is $1e-4$ ($1e-3$ until September 2026): the predicted gain is itself an estimate, and at $1e-3$ the block-diagonal metric ended $1.1e-3$ to $1.3e-3$ nats short of the optimum in 6 of 18 large fits; at $1e-4$, none of 18 ended more than $2e-4$ short, for 3 to 5 per cent more time.
laplaceGradTol	Numeric. Tolerance for the item-block gradient, measured relative to the objective rather than as a raw norm: the reported norm is divided by the same denominator <code>laplaceTolScale</code> selects for the objective. An absolute norm grows with both the number of item parameters and the number of responses, so an absolute threshold cannot be met by a large sparse fit however well converged it is. Both the raw and the scaled value are returned in <code>laplaceStatus</code> . Formerly a gradient-norm tolerance for the item-step Laplace surrogate. Default is $1e-4$.
laplaceTolScale	Character. What the scaled gradient norm compared against <code>laplaceGradTol</code> is measured against on the Laplace path (<code>laplaceTol</code> itself is the separate, unscaled predicted-gain criterion under <code>laplaceOptimizer = "fisher"</code>). <code>"relative"</code> (the default, and the historical behaviour) divides the raw gradient norm by the size of the objective. Because the objective grows with the number of responses, the same nominal tolerance becomes laxer as data grow: on several million responses a relative change of $1e-3$ is an absolute change in the thousands, and a fit can stop while it is still moving. <code>"per_obs"</code> divides by the number of responses instead, so the quantity thresholded is the gradient per response and does not depend on data size. Prefer it when comparing fits across designs of very different sizes, but note that it is far stricter at large N : a tolerance chosen for <code>"relative"</code> should not be carried over unchanged.
laplacePolish	Integer. Maximum block-Newton polish steps taken after the quasi-Newton optimizer stops, on the final fit only. L-BFGS gives up early on large problems: at eight million responses it reported no further progress while a single Newton step still recovered 146 units of objective. The curvature is the per-item information already used for standard errors, so a step costs one closed-form assembly and a short backtracking search. Applies to <code>laplaceOptimizer = "mize"</code> only – the Fisher-shaped optimizer has no polish stage, because it already reaches the optimum L-BFGS gives up short of. Default is NA, meaning the optimiser's own choice: 8 under <code>"mize"</code> , none under <code>"fisher"</code> . Set to 0 to disable under <code>"mize"</code> ; an explicit nonzero value under <code>"fisher"</code> is refused rather than silently ignored.
laplacePersonTol	Numeric. Newton tolerance for person-mode updates in <code>"laplace"</code> : the gradient norm at which each person's mode solve stops. The objective is a sum over persons, and every mode is solved from where the previous evaluation left it, so this sets how much the value depends on the path that led to it. At $1e-4$ the same point evaluated from different warm starts differed by about $1e-3$ nats at 5,000 persons and $1e-2$ at 40,000 – above the optimiser's own stopping tolerance (<code>laplaceTol</code>). At $1e-6$ it is about $1e-5$ and $1e-4$, for 3 to 5 per cent more time per fit. Default is $1e-6$ ($1e-4$ before September 2026).

`laplaceCertify` Logical, character or numeric. How to certify convergence on the Laplace path, on the curvature of the objective the fit ended on (Laplace, or the refined marginal). The optimiser's own stopping rule is a limited-memory estimate of the same quantity and cannot see a transform that has gone flat, where the gradient and the curvature vanish together (a correlation parked at 1.000 with a gradient of exactly zero); the certificate can.

The fit is certified when the predicted gain $\frac{1}{2}g'H^{-1}g$ is below ten times `laplaceTol` nats, no direction has negative curvature, and stepping along the directions with no curvature gains nothing; a discrimination, asymptote or latent correlation whose transform has gone flat is pulled back towards its live range, and a fit that gains from that is not a maximum. An item whose response function has run off – a loading steeper than `getOption("bigIRT.runaway_slope", 20)` logits per ability SD, or an intercept that many logits beyond every ability within four SDs, which unregularised fits (`priors = FALSE`) can reach – makes the verdict "diverged": its parameters are named in the reason and in `laplaceStatus$diverged` (item, parameter, value, and what pulling the item back along its own ray gained), and `converged` is `FALSE`. That table is filled, and the reason set to "diverged", even when the certificate is off. `converged` follows the verdict, `laplaceStatus$certificate` holds it, and the default standard errors of `itemInformation`, `summary()` and `tidyIRT()` come from its curvature.

`TRUE` or "fast" (the default): each item's block of the Hessian in one quadrature pass, its own covariate coefficients (`itemSpecificBetas`) included, the parameters shared across items by differencing, items coupled through those, which drops only the coupling between two items through the persons who answered both. It takes the time of about five to seven evaluations whatever the number of items or persons, a few per cent of a fit. On a fit that was not refined (`laplaceRefine = "none"`, or a 2PL at one dimension under "auto") this curvature is the quadrature marginal's at the fitted modes rather than the Laplace objective's; the two agree near the optimum, and far from it a fit the exact route calls "suboptimal" may be called "notmaximum" here – neither certifies it. Because this route can also find negative curvature the exact one does not, along a direction only a prior identifies, a "notmaximum" from negative curvature is re-tested by the exact route whenever that projects within `getOption("bigIRT.certify_escalate", 60)` seconds; the exact verdict is then the one reported, and the fast one is kept in `laplaceStatus$certificate$fast`.

"exact": the whole Hessian by central differences of the gradient, two evaluations per free parameter, which roughly doubles a small fit and is out of reach for large ones. Ask for it to have the fitted objective's own curvature, with the coupling between items kept, when a small fit's standard errors or an unrefined fit's verdict are what will be reported. A number is the exact route with a budget in seconds, skipped with a message when one timed evaluation projects past it. `FALSE` turns the certificate off: `converged` is then the optimiser's own verdict and the default standard errors the closed-form profile ones. Default `TRUE`.

`laplaceJitter` Numeric. Small jitter added to stabilise the person precision factorisations. Default is `1e-6`.

`laplaceKeepCovariance`

Logical. Whether to keep each person's posterior covariance matrix (`fit$personPosterior$covariance`

which `reliability` and `looIRT` need. The precision is always kept. Covariances are otherwise retained only when the fit needs them (estimated correlations, empirical Bayes). Default is FALSE.

<code>laplaceDiagnostics</code>	Logical. Whether to store per-evaluation optimiser diagnostics in <code>fit\$laplaceDiagnostics</code> . Default is FALSE.
<code>laplacePlot</code>	Logical. Whether to draw the optimiser diagnostic plot during fitting. Default is FALSE.
<code>laplacePlotEvery</code>	Integer. Plot every N outer iterations when <code>laplacePlot=TRUE</code> . Default is 1.
<code>normalise</code>	Logical. Whether to normalize the output estimates. Default is FALSE.
<code>normaliseScale</code>	Numeric. Scale for normalization. Default is 1.
<code>normaliseMean</code>	Numeric. Mean for normalization. Default is 0.
<code>dropPerfectScores</code>	Logical. Whether to drop perfect scores from each subject and item before estimation. Default is TRUE.
<code>trainingRows</code>	Integer vector. Rows of data to use for estimation of parameters. Default is all rows in <code>dat</code> . Every item must keep at least one training response, since there is nothing else from which to calibrate it. Persons need not: a person whose responses are all excluded has the prior as their posterior, so their fitted ability is exactly the prediction from their <code>personPreds</code> (or the prior mean if there are none). Holding out whole persons this way is the design that answers whether person covariates say anything about somebody the model has never seen – excluding only some of a person’s responses cannot, because their own ability parameter is still estimated from the rest.
<code>cores</code>	Integer. Threads for the person-level kernels. Default is 6.
<code>verbose</code>	Integer. 0 is silent; 1 reports the start and end of the fit and each empirical-Bayes round; 2 adds a line per objective evaluation and the optimiser’s timing summary. Default is 0.
<code>keepInternals</code>	Logical. Attach the prepared data (<code>sdat</code>), the final parameter state and the prior precision array to the returned fit. Off by default because these carry copies of the response arrays. Useful for verifying analytic gradients against finite differences, and for inspecting a fit that stopped somewhere unexpected.
<code>...</code>	Accepted only for arguments of earlier versions, so that old code keeps running. <code>plot</code> and <code>noptimsteps</code> , which never changed a result, are ignored in silence. <code>iter</code> , <code>tol</code> , <code>init</code> , <code>carefulfit</code> and <code>stochastic</code> configured the Stan joint-MAP path removed in 0.2.0 (the optimiser’s own controls are <code>laplaceOuterIter</code> and <code>laplaceTol</code>), and <code>laplaceItemIter</code> and <code>ebayesFromFixed</code> had no effect; these are ignored with a warning. Any other argument is an error.

Details

"laplace" (default) Marginal maximum likelihood (or posterior mode, with priors): the abilities are integrated out by a Laplace approximation at each person’s mode, refined by adaptive quadrature where Laplace is biased (`laplaceRefine`).

"map" The joint posterior mode over item parameters and abilities together. Each evaluation solves every person's mode and differentiates the joint posterior at it, so it is cheaper per evaluation than Laplace, and much cheaper than Laplace refined by quadrature. Its latent scale is set by the priors rather than the data (see `marginalApprox`).

Both run the same optimiser over the item-side parameters with every person mode solved inside each evaluation, so the optimiser controls (`laplaceOptimizer`, `laplaceTol`, `laplaceOuterIter`, `laplacePersonTol`, `laplaceCertify`, ...) apply to either; the `laplace` prefix is historical.

Value

An object of class `bigIRT_fit`, the same under either estimator. Its main parts:

`itemPars`, `personPars` Item parameters (one row per item, with a column per loading in more than one dimension) and person abilities with their posterior SDs (`_SD` columns).

`pars` Everything estimated, including the free parameters on their unconstrained scales and the row-level effective parameters.

`personPosterior` Each person's mode and the precision (and, with `laplaceKeepCovariance`, the covariance) of their posterior, conditional on the item parameters.

`abilityPrior` The ability prior's SDs, correlation (estimated or fixed; see `estimated_corr`) and precision.

`optim` The optimiser's result: `logLik` is the fitted objective – the penalised Laplace (or refined marginal, or the joint log posterior for "map" – and `grad` and `layout` its gradient by parameter block. Use `logLik()` for a likelihood.

`laplaceStatus` Convergence and its diagnosis, for either estimator (the name is historical): converged and reason, the convergence certificate, run-off items, and iteration-limit, person-mode-failure, numerical-failure, frozen-effect, approximate-gradient and covariance-retention flags. `estimator` and `marginal` say which objective was fitted; for "map", `marginal_logLik` is the marginal likelihood at the MAP estimates.

`laplaceDiagnostics` When requested, one row per objective evaluation: objective, relative improvement, item and person RMS steps, item gradient, posterior SD, person convergence, timings, optimiser work and the strict criterion.

`prepared` The prepared data the backend consumed: indices, response vectors, fixed-value maps and prior hyperparameters.

`backend` "laplace" or "map".

The frozen-effect flags, `beta_frozen` and its alias `frozen_effects`, are TRUE only when `personPreds` were supplied *and* their ability coefficients were held out of the optimised parameter vector. Both estimators always estimate them, so these read FALSE on any fit that has person predictors; a fit with none reports FALSE too, because there is nothing that could be frozen. (Before this was corrected the flag was set from the presence of person predictors alone, so it was TRUE for precisely the fits that estimate the coefficients, and `print()` wrongly announced that the effects had been held fixed.)

Row ordering. `fitIRT` sorts the data by person before fitting, so most row-level objects are in that internal order. The two per-response predictions users normally want, `fit$pars$p` (the probability of the observed response) and `fit$pars$pcorrect` (the probability of a correct response), are mapped back to the order of the data passed in, so they can be indexed with the same row

numbers used for trainingRows; rows removed by dropPerfectScores are NA. Every other row-level object – b_row, c_row, d_row, eta_row, row_loadings, row_ability, and the contents of fit\$prepared – remains in internal order, because the person-covariance routines pair those with fit\$prepared\$id. Use fit\$pars\$originalRow, which gives the input row index of each internal row, to move between the two.

Examples

```
#Generate some data (here 2pl model
require(data.table)
dat <- simIRT(Nsubs = 50, Nitems = 100, Nscales = 2,
  logitCMean = -10, logitCSD = 0, AMean = 1, ASD = .3,
  BMean=0, BSD = .5,
  AbilityMean = 0, AbilitySD = 1)

#Fit by marginal maximum likelihood (the default), which integrates the
#abilities out by a Laplace approximation.
fit <- fitIRT(dat$dat, cores=2, score = 'score', id = 'id',
  scale = 'Scale', item = 'Item', pl=2)

print(fit$personPars)
print(fit$itemPars)

#The same model by the joint posterior mode: same arguments, same output.
fit_map <- fitIRT(dat$dat, cores = 2, pl = 2, marginalApprox = 'map')
logLik(fit_map) # the model's marginal likelihood at the MAP estimates

#Explicit multidimensional loading specification (2D example)
#Rows/cols are aligned by names to item ids and scale ids.
item_ids <- as.character(unique(dat$dat$Item))
scale_ids <- as.character(unique(dat$dat$Scale))
L <- matrix(NA_real_, nrow = length(item_ids), ncol = length(scale_ids),
  dimnames = list(item_ids, scale_ids))
L[1, ] <- c(0.8, 0.2) #fixed cross-loading
L[2, 2] <- 0 #fixed zero loading
fit_mirt <- fitIRT(dat$dat, pl = 2, cores = 1, loadings = L)
head(fit_mirt$itemPars[, grep("^A(_|$)", names(fit_mirt$itemPars)), drop = FALSE])
```

glanceIRT

One-row model summary

Description

Registered against broom: : glance() when broom is installed.

Usage

```
glanceIRT(x, ...)
```

Arguments

`x` A fitted object from `fitIRT()`.
`...` Unused.

Value

A one-row data frame of backend, log-likelihood, degrees of freedom, AIC, BIC, sample sizes, convergence, reliability and held-out log loss.

heldoutLogLik	<i>Held-out marginal log-likelihood</i>
---------------	---

Description

The marginal log-likelihood of the persons a fit held out entirely (none of their responses in `trainingRows`), each over all of their responses, at the fitted parameters: the criterion `improveModel()` uses to compare models. It integrates each person's ability by the fit's quadrature rule around their posterior mode, and includes the ability prior's normalising constant, so models with different numbers of dimensions or ability variances compare correctly.

Usage

```
heldoutLogLik(fit, cores = 1L, nodes = NULL)
```

Arguments

`fit` A fit from `fitIRT()` with whole persons held out of `trainingRows`.
`cores, nodes` As for `scoreContributions()`.

Value

A list: `loglik`, `persons`, `responses`; `NULL` when no person was held out entirely.

heldoutMetrics	<i>Predictive performance on held-out responses</i>
----------------	---

Description

Scores the rows excluded by `trainingRows` against the item base rate, which predicts each item's training success rate and ignores the person. That is the fitted model with zero discrimination, so it lies inside the model's own parameter space and is a fair floor.

Usage

```
heldoutMetrics(fit, eps = 1e-12)
```

Arguments

<code>fit</code>	A fitted object from <code>fitIRT()</code> , fitted with <code>trainingRows</code> .
<code>eps</code>	Numeric. Probability clamp for the log loss.

Value

A list of held-out response count, log loss, Brier score, AUC, accuracy and the item-base-rate Brier score, or NULL when nothing was held out.

<code>improveModel</code>	<i>Improve a model by score tests and held-out likelihood</i>
---------------------------	---

Description

Starting from a base model, repeatedly tests candidate changes at the current fit (`modificationTests()`), refits the promising ones, and keeps a change only if it improves the marginal likelihood of persons held out of the fit entirely (`heldoutLogLik()`). Changes come in four kinds, each refitted as a batch: covariates of ability (indicators, or a 3-df natural spline), DIF on item intercepts or slopes for the flagged items along person or response covariates, freed lower or upper asymptotes for the flagged items, and group dimensions for item sets with local dependence (each set's items load on a dimension of its own, uncorrelated with the rest). A candidate must be significant after Benjamini-Hochberg within its kind and large enough to matter (`minEffect` change in the probability of a correct answer for DIF, `minAsymptote` for an asymptote, `minGroupSD` for a group dimension's SD on the logit scale); a batch must raise the held-out log-likelihood by at least `minGain`. Accepted batches are combined and the next round tests the combined model. The final model is refitted on all persons.

Usage

```
improveModel(
  dat,
  score = "score",
  id = "id",
  item = "Item",
  scale = "Scale",
  pl = 2L,
  personPreds = character(),
  personCovariates = character(),
  responseCovariates = character(),
  itemSets = character(),
  dif = c("B", "A"),
  asymptotes = c("lower", "upper"),
  holdout = 0.2,
  alpha = 0.05,
  minEffect = 0.03,
  minAsymptote = 0.05,
```

```

minGroupSD = 0.3,
minZ = stats::qnorm(0.95),
minGain = 0,
nsimDim = 19L,
maxRounds = 3L,
maxGroups = 2L,
seed = 1L,
cores = 1L,
verbose = TRUE,
fitArgs = list(laplaceRefine = "agq")
)

```

Arguments

<code>dat</code>	Long data, one row per response.
<code>score, id, item, scale, pl</code>	As for <code>fitIRT()</code> ; the base model.
<code>personPreds</code>	Person predictors in the base model's ability regression.
<code>personCovariates, responseCovariates, itemSets, dif, asymptotes</code>	What to test; see <code>modificationTests()</code> .
<code>holdout</code>	Share of persons held out to judge changes.
<code>alpha</code>	FDR level for candidates.
<code>minEffect, minAsymptote, minGroupSD</code>	Smallest changes worth adding.
<code>minZ, minGain</code>	A batch is kept when the held-out persons' log-likelihood rises, compared person by person: by at least <code>minGain</code> nats in total and by at least <code>minZ</code> standard errors of the per-person differences (default the one-sided 5 per cent point). In simulation (50 replications per truth) this found 80 per cent of DIF items and the testlet in 88 to 94 per cent of runs, where $z = 2$ found 70 and 72 to 84, and with no true change it added nothing at either.
<code>nsimDim</code>	Simulations for the null of the exploratory dimensions (0 turns them off); see <code>modificationTests()</code> .
<code>maxRounds, maxGroups</code>	Limits on rounds and on group dimensions.
<code>seed</code>	For the holdout split.
<code>cores</code>	Threads.
<code>verbose</code>	Print progress.
<code>fitArgs</code>	Further <code>fitIRT()</code> arguments (default <code>laplaceRefine = "agq"</code>).

Details

Selection uses the data, so the final model's estimates and standard errors are conditional on it: the held-out gain is the honest measure of what the changes are worth, not the in-sample fit.

Value

A bigIRT_improved object: fit (final model, all persons), spec (the terms added), path (each batch tried: round, kind, terms, held-out log-likelihood before and after, accepted), screens (the last round's tests), base_fit (the base model on the training persons).

See Also

[modificationTests\(\)](#), [heldoutLogLik\(\)](#)

 IRTcurve

Evaluate IRT Item Response Curves

Description

Evaluates 1D or multidimensional 4PL response curves under the bigIRT parameterization

$$\eta = \theta A^{\top} - B.$$

Usage

```
IRTcurve(
  A,
  B,
  C = 0,
  D = 1,
  theta = seq(-3, 3, 0.01),
  plot = TRUE,
  rescale = FALSE,
  add = FALSE,
  item = NULL,
  ...
)
```

Arguments

A	Numeric item-by-factor loading matrix, or a numeric vector for a single-factor model.
B	Numeric vector of item intercept/difficulty parameters.
C	Numeric vector of lower asymptotes. Defaults to 0.
D	Numeric vector of upper asymptotes. Defaults to 1.
theta	Numeric matrix of latent trait values (rows are evaluation points, columns are factors), or a numeric vector for a single-factor model.
plot	Logical; if TRUE, produce a 1D plot.
rescale	Logical; if TRUE, standardize theta before evaluation.
add	Logical; when plotting 1D curves, add to an existing plot.
item	Optional item index/indices to evaluate.
...	Additional graphical arguments passed to plotting functions.

Value

A numeric vector for a single item or a numeric matrix with rows corresponding to theta points and columns to items.

 IRTic

Information criteria for a marginal fit

Description

Person parameters are integrated out of the marginal likelihood, so they are not free parameters of it and are not counted; counting them would penalise a model for the size of its sample. For a joint-MAP fit the likelihood is that same marginal, evaluated at the MAP estimates (see [logLik.bigIRT_fit\(\)](#)), so the criteria compare fits by either estimator on one scale. Effective and nominal degrees of freedom are both reported, as are BIC on persons and on responses, since the two answer different questions.

Usage

```
IRTic(fit, cores = 1L)
```

Arguments

fit	A fitted object from fitIRT() .
cores	Integer. Cores for the effective-parameter calculation.

Value

A data frame of criterion, degrees-of-freedom type, sample-size type, degrees of freedom and value.

 itemFit

Item fit by standardised residuals

Description

Outfit is the mean squared standardised residual, and so is pulled about by unexpected responses far from an item's difficulty; infit weights by information and is the more stable of the two. Both are one pass over the responses.

Usage

```
itemFit(fit, training_only = TRUE)
```

Arguments

fit	A fitted object from fitIRT() .
training_only	Logical. Score training rows only.

Value

A data frame of item, response count, infit, outfit and a standardised outfit.

Note

These are Rasch-family statistics. Their reference distributions are only well behaved for one- and two-parameter models, and should not be read as calibrated tests under a three- or four-parameter fit.

See Also

[empiricalICC\(\)](#)

itemInformation

Item information, standard errors and effective parameters

Description

Observed information for each item's free parameters, with the standard errors and effective parameter count that follow from it. With `priors = TRUE` the fit is penalised, so these describe posterior curvature rather than repeated sampling.

Usage

```
itemInformation(
  fit,
  cores = 1L,
  method = c("auto", "exact", "profile", "hessian"),
  person_tol = 1e-08,
  eps = 1e-04,
  max_blocks = 2000L,
  time_budget = 60
)
```

Arguments

<code>fit</code>	A fitted object from fitIRT() , fitted with <code>keepInternals = TRUE</code> (not needed for "exact" when the certificate kept the covariance).
<code>cores</code>	Integer. Cores for the objective evaluations.
<code>method</code>	Character. "exact" takes each item's block of the full inverse of the exact Hessian of the fitted objective – the refined marginal when <code>laplaceRefine</code> ran – so every standard error allows for the other items and the shared parameters being estimated too; it is read from the fit's convergence certificate when that kept it (see <code>laplaceCertify</code> in fitIRT()), which costs nothing and needs no internals, and computed otherwise. "auto" takes "exact" when the certificate kept it, and otherwise times one evaluation and takes "hessian" when it fits

	time_budget. "hessian" differences the analytic gradient per item block and inverts each block alone, which holds everything else at its estimate. "profile" uses a closed-form weight and costs nothing, but its standard errors run about six per cent small.
person_tol	Numeric. Person-mode tolerance for the differenced method. Must be far tighter than the fitting default, or the difference quotient measures frozen-mode curvature instead.
eps	Numeric. Step size for the difference quotients.
max_blocks	Integer. Refuse "hessian" above this many items.
time_budget	Numeric. Seconds "auto" may spend before falling back.

Value

An object of class bigIRT_iteminfo. P counts the parameter types free on at least one item; a parameter an item has fixed (an anchor through itemDat) is NA in every per-item entry below, and the item's block covers only the parameters it has free.

information P x P x Nitems array of likelihood information; for "exact", the inverse of each item's covariance block, prior included.

prior_precision Matching prior precision, zero without priors or where information already carries it.

covariance Inverse of each block; for "exact", each item's block of the full inverse.

se Nitems x P matrix of standard errors.

edf Effective parameters per item and block.

free Nitems x P logical matrix, TRUE where the item has that parameter free.

se_loadings For "exact" on a fit with signed loadings (signedLoadings in `fitIRT()`) only: the standard error of every free loading's parameter, Nitems x Nscales, NA where the loading is fixed. With cross-loadings free there is more than one loading per item, which the per-item blocks do not describe.

See Also

[checkConvergence\(\)](#)

logLik.bigIRT_fit	<i>Marginal log-likelihood of a fit</i>
-------------------	---

Description

The log-likelihood with the abilities integrated out, at the fitted item parameters, with the item prior density removed and the ability prior's normalising constants restored. For a joint-MAP fit (marginalApprox = "map"), the marginal likelihood of the same model evaluated at the MAP estimates, by the approximation laplaceRefine selects, so that fits by either estimator are compared on one scale.

Usage

```
## S3 method for class 'bigIRT_fit'
logLik(object, effective = TRUE, cores = 1L, ...)
```

Arguments

object	A fitted object from fitIRT() .
effective	Logical. Report effective rather than nominal degrees of freedom when the fit is penalised.
cores	Integer. Cores for the effective-parameter calculation.
...	Unused.

Value

An object of class `logLik`, with attributes `df`, `nominal_df`, `nobs` (persons) and `penalised`.

looIRT	<i>Approximate leave-one-out cross-validation</i>
--------	---

Description

The Laplace backend leaves a Gaussian posterior for each person, so draws from it can be importance-weighted and handed to `loo`. Item parameters stay at their mode, so the result is conditional on them; that is mild when persons greatly outnumber the responses per item. Pareto k is the check that matters, and held-out cross-validation is the better tool when it is large.

Usage

```
looIRT(fit, draws = 500L, persons = NULL, seed = NULL, cores = 1L)
```

Arguments

fit	A fitted object from fitIRT() , fitted with <code>laplaceKeepCovariance = TRUE</code> .
draws	Integer. Posterior draws per person.
persons	Integer. Optional person subsample, for datasets too large to hold a draws-by-responses matrix.
seed	Integer. Optional seed for the draws.
cores	Integer. Passed to <code>loo::loo()</code> .

Value

An object of class `loo`.

See Also

[heldoutMetrics\(\)](#)

modificationTests	<i>Score tests for modifying a fitted model</i>
-------------------	---

Description

Tests candidate changes to a fit at the fit itself, without fitting any alternative: covariates of ability, DIF on item intercepts or slopes along person or response covariates, per-item lower and upper asymptotes, and group dimensions shared by sets of items. Each comes with a one-step estimate of the change.

Usage

```
modificationTests(
  fit,
  dat,
  personCovariates = character(),
  responseCovariates = character(),
  itemSets = character(),
  dif = c("B", "A"),
  asymptotes = c("lower", "upper"),
  purify = TRUE,
  exploratory = TRUE,
  nsimDim = 19L,
  minSetItems = 3L,
  cores = 1L
)
```

Arguments

fit	A fit from <code>fitIRT()</code> .
dat	The data the fit was made from (for covariates and item sets).
personCovariates	Columns of dat constant within person: tested as covariates of ability, and for DIF.
responseCovariates	Columns of dat varying by response: tested for DIF.
itemSets	Columns of dat constant within item; each level with at least minSetItems items is a candidate group dimension.
dif	Item parameters tested for DIF: "B" (intercept), "A" (slope), or none.
asymptotes	"lower", "upper", both or none.
purify	Purify the DIF tests (recommended: it sets the anchors).
minSetItems	Smallest item set tested for a group dimension.
cores	Threads.

Details

Ability covariates and DIF use `scoreTest()` (DIF purified, so it is measured against anchors, with effect sizes). An asymptote fixed at c (or d) is tested against freeing it: P is linear in c, so a response's score is exact from its leave-one-out probability. A set of items is tested for a shared person effect $u \sim N(0, \tau^2)$ on their linear predictors, the variance-component score test (Lin, 1997) from each person's posterior expectation of $(\sum g)^2 + \sum h$ over the set; a significant one is local dependence among the set's items, which a group dimension models. Asymptote and group tests are one-sided.

Value

A list of data.tables: ability, dif (with effect sizes), asymptotes, groups; each with p-values adjusted by Benjamini-Hochberg (`p_fdr`) within its kind.

See Also

`improveModel()`, `scoreTest()`

normaliseIRT	<i>normaliseIRT</i>
--------------	---------------------

Description

Normalise item response theory (IRT) parameters.

Usage

```
normaliseIRT(  
  B,  
  Ability,  
  A,  
  normbase = "Ability",  
  normaliseScale = 1,  
  normaliseMean = 0,  
  robust = TRUE,  
  AbilitySD = NULL,  
  AbilityMean = NULL  
)
```

Arguments

- | | |
|----------|--|
| B | Vector of item difficulty parameters. |
| Ability | Vector of persons' ability parameters. |
| A | Vector of item discrimination parameters. |
| normbase | The base from which the normalisation should be calculated. Can be 'Ability' or 'B'. |

`normaliseScale` The scale to normalise to.

`normaliseMean` The mean to normalise to. The default is 0 for the supported normbase values ('Ability' or 'B').

`robust` If TRUE, the location and scale come from the median and the interquartile range, the latter converted to a standard deviation (divided by 1.349, its value for a normal distribution). Until October 2026 the interquartile range itself was set to `normaliseScale`, which left a normal latent variable with standard deviation .74.

`AbilitySD, AbilityMean` Optional latent standard deviation and mean of the solution, used in place of the spread and centre of `Ability` when `normbase = 'Ability'`. Supply them when `Ability` holds point estimates (MAP, EAP): those are shrunk toward the prior mean, so their spread understates the latent scale by roughly the square root of the reliability, and normalising by it flattens the discriminations by the same factor. For a `fitIRT` fit, `extractMIRTpars()` returns them as `latent_sd` and `latent_mean`.

Value

A list containing the normalised A, B, and Ability parameters, and for `normbase 'Ability' or 'B'` the center and scale of the transform: the normalised abilities are $(\text{Ability} - \text{center}) / \text{scale} + \text{normaliseMean}$.

Examples

```
B <- rnorm(100,2,1)
Ability <- rnorm(500,3,.5)
A <- rnorm(100,1.4,.05)
normaliseIRT(B, Ability, A, normbase='B')
```

normaliseMIRT

Normalize Multidimensional IRT Parameters

Description

Applies a curve-preserving affine transform to a multidimensional IRT solution so that the latent ability distribution is centered and whitened, with optional orthogonal alignment to a reference loading matrix.

Usage

```
normaliseMIRT(
  B,
  Ability,
  A,
```

```

    AbilityCorr = NULL,
    normaliseScale = 1,
    normaliseMean = 0,
    jitter = 1e-08,
    referenceA = NULL,
    align = c("none", "orthogonal"),
    AbilitySD = NULL,
    AbilityMean = NULL
  )

```

Arguments

B	Numeric vector of item intercept/difficulty parameters.
Ability	Numeric matrix of person abilities with rows corresponding to persons and columns to latent factors. Its mean and covariance define the transform, so it should represent the latent distribution: true or generating abilities, or draws from the fitted posterior. Point estimates (MAP, EAP) are shrunk toward the prior mean, so their spread understates the latent scale by roughly the square root of the reliability, and the loadings come out flattened by the same factor – about a fifth at ten responses per person. Solutions that already share an identification (the same fixed latent distribution and a confirmatory loading pattern) need no transform at all.
A	Numeric item-by-factor loading matrix.
AbilityCorr	Optional latent correlation matrix. If omitted, the correlation is estimated from Ability.
normaliseScale	Numeric scalar giving the target latent SD after whitening. Defaults to 1.
normaliseMean	Numeric scalar or vector giving the target latent mean after normalization. Defaults to 0.
jitter	Small positive constant added for numerical stability.
referenceA	Optional reference loading matrix used for orthogonal Procrustes alignment.
align	Character string, either "none" or "orthogonal".
AbilitySD, AbilityMean	Optional latent standard deviations and means, one per factor, used in place of the spread and centre of Ability. Pass them whenever Ability holds point estimates; <code>extractMIRTpars()</code> returns them for bigIRT and mirt fits as <code>latent_sd</code> and <code>latent_mean</code> .

Value

A list containing normalized A, B, Ability, and associated affine transform pieces (center, chol_cov, inv_chol, AbilityCorr, sign, and rotation).

personCovarianceMatrices

Compute person covariance matrices from row-level moderated IRT parameters

Description

Builds a posterior precision matrix for each person by combining a prior precision matrix with row-wise Fisher information contributions. The intended input is the row-level moderated parameter state a fit reports (`fit$rowEffective`):

- a current person mean matrix `theta_mean`
- row-level effective difficulties `b`
- row-level effective lower and upper asymptotes `c` and `d`
- row-level effective loading vectors, supplied either densely or in sparse row form

Usage

```
personCovarianceMatrices(
  id,
  theta_mean,
  b,
  loadings,
  c = NULL,
  d = NULL,
  prior_precision = NULL,
  jitter = 1e-08,
  max_attempts = 8,
  return_precision = TRUE
)
```

Arguments

<code>id</code>	A vector of person ids with one entry per observation row.
<code>theta_mean</code>	A numeric $N_{subs} \times K$ matrix of current person means. Rows must correspond to the sorted unique values of <code>id</code> .
<code>b</code>	Numeric vector of row-level effective difficulties.
<code>loadings</code>	Either an $N_{obs} \times K$ numeric matrix of row-level effective loadings, or a sparse list returned by rowLoadingsToSparse .
<code>c</code>	Numeric vector of row-level lower asymptotes. Defaults to all zeros.
<code>d</code>	Numeric vector of row-level upper asymptotes. Defaults to all ones.
<code>prior_precision</code>	Either <code>NULL</code> , a shared $K \times K$ prior precision matrix, or a person-specific $K \times K \times N_{subs}$ array.

jitter	Initial diagonal jitter added before Cholesky factorisation.
max_attempts	Maximum number of jitter expansions before failing.
return_precision	Logical. If TRUE, also return posterior precision and precision Cholesky arrays. If FALSE, compute and return only covariance (plus jitter diagnostics).

Details

For each response row r attached to person j , the function computes

$$Q_j = Q_{0,j} + \sum_r w_r a_r a_r^\top$$

where $Q_{0,j}$ is the prior precision for person j , a_r is the row-level effective loading vector, and

$$w_r = \frac{(p'_r(\eta_r))^2}{p_r(\eta_r)(1 - p_r(\eta_r))}$$

is the Bernoulli Fisher-information weight. This is stable for 2PL, 3PL, 4PL, and sparse multidimensional loadings.

The embedded C++ backend returns the posterior precision, its lower Cholesky factor, and the implied covariance matrix for each person. The pure R version `personCovarianceMatrices_R` mirrors the same calculations for sanity checks.

Value

A list with:

- covariance Posterior covariance array with dimensions $K \times K \times N_{\text{subs}}$.
- precision Posterior precision array with dimensions $K \times K \times N_{\text{subs}}$.
- precision_chol Lower Cholesky factor of the posterior precision for each person.
- chol_jitter_used The diagonal jitter needed for each person-specific factorisation.
- id_levels Unique person ids corresponding to the third array dimension.
- backend Either "cpp" or "R".

Examples

```
# Two-dimensional row-level example with one cross-loading row.
theta_mean <- rbind(
  c(-0.3, 0.4),
  c(0.6, -0.2)
)
colnames(theta_mean) <- c("math", "verbal")

row_loadings <- rbind(
  c(1.2, 0.0),
  c(0.8, 0.3),
  c(0.0, 1.1),
  c(0.5, 0.6)
```

```

)

ids <- c("p1", "p1", "p2", "p2")
b <- c(-0.5, 0.1, 0.0, -0.2)
cpar <- rep(0, length(ids))
dpar <- rep(1, length(ids))

sparse_loadings <- rowLoadingsToSparse(row_loadings)
out_cpp <- personCovarianceMatrices(
  id = ids,
  theta_mean = theta_mean,
  b = b,
  c = cpar,
  d = dpar,
  loadings = sparse_loadings,
  prior_precision = diag(2)
)

out_r <- personCovarianceMatrices_R(
  id = ids,
  theta_mean = theta_mean,
  b = b,
  c = cpar,
  d = dpar,
  loadings = sparse_loadings,
  prior_precision = diag(2)
)

max(abs(out_cpp$covariance - out_r$covariance))
out_cpp$covariance[, , "p1"]

```

personCovarianceMatrices_R

Compute person covariance matrices with a pure R reference implementation

Description

This function mirrors [personCovarianceMatrices](#) but performs the row-wise precision accumulation and per-person matrix inversions in R. It is intended for sanity checks against the embedded C++ implementation.

Usage

```

personCovarianceMatrices_R(
  id,
  theta_mean,
  b,
  loadings,

```

```

    c = NULL,
    d = NULL,
    prior_precision = NULL,
    jitter = 1e-08,
    max_attempts = 8,
    return_precision = TRUE
  )

```

Arguments

id	A vector of person ids with one entry per observation row.
theta_mean	A numeric $N_{subs} \times K$ matrix of current person means. Rows must correspond to the sorted unique values of id.
b	Numeric vector of row-level effective difficulties.
loadings	Either an $N_{obs} \times K$ numeric matrix of row-level effective loadings, or a sparse list returned by rowLoadingsToSparse .
c	Numeric vector of row-level lower asymptotes. Defaults to all zeros.
d	Numeric vector of row-level upper asymptotes. Defaults to all ones.
prior_precision	Either NULL, a shared $K \times K$ prior precision matrix, or a person-specific $K \times K \times N_{subs}$ array.
jitter	Initial diagonal jitter added before Cholesky factorisation.
max_attempts	Maximum number of jitter expansions before failing.
return_precision	Logical. If TRUE, also return posterior precision and precision Cholesky arrays. If FALSE, compute and return only covariance (plus jitter diagnostics).

Value

A list with posterior covariance; and if return_precision=TRUE, also precision and precision_chol arrays.

plotLaplaceDiagnostics

Plot Laplace diagnostics

Description

Plot Laplace diagnostics

Usage

```
plotLaplaceDiagnostics(fit, logGrad = TRUE, showTiming = TRUE)
```

Arguments

fit	A fitted bigIRT model returned by <code>fitIRT()</code> with either Laplace backend and <code>laplaceDiagnostics=TRUE</code> . Direct-Laplace fits do not have blockwise stability panels; those panels are shown as unavailable rather than interpreted as failures.
logGrad	Whether to plot the item-step gradient norm on a log10 scale.
showTiming	Whether to include timing and stability panels.

Value

Invisibly returns the diagnostic data frame used for plotting.

reliability	<i>Reliability of the fitted abilities</i>
-------------	--

Description

Empirical reliability compares the spread of the estimates against that spread plus the average error variance; marginal reliability compares the average error variance against the prior variance. They disagree when the estimates are shrunk, so both are reported.

Usage

```
reliability(fit)
```

Arguments

fit	A fitted object from <code>fitIRT()</code> , fitted with <code>laplaceKeepCovariance = TRUE</code> .
-----	--

Value

A data frame with one row per latent dimension.

reportIRT	<i>Render a model report</i>
-----------	------------------------------

Description

A self-contained HTML report: convergence, the item table, information and reliability, person and item coverage, item fit, the worst-fitting response curves, and held-out performance. Everything in it comes from the exported functions, so it is a convenience rather than a second implementation.

Usage

```
reportIRT(
  fit,
  file = "bigIRT-report.html",
  title = "bigIRT model report",
  method = "profile",
  max_icc = 9L,
  cores = 1L,
  quiet = TRUE
)
```

Arguments

fit	A fitted object from <code>fitIRT()</code> , fitted with <code>keepInternals = TRUE</code> .
file	Character. Output path.
title	Character. Report title.
method	Character. Passed to <code>itemInformation()</code> .
max_icc	Integer. Worst-fitting items to draw curves for.
cores	Integer. Cores for the information calculation.
quiet	Logical. Passed to <code>rmarkdown::render()</code> .

Value

The output path, invisibly.

rowLoadingsToSparse	<i>Convert row loadings to a sparse row representation</i>
---------------------	--

Description

Converts an $N_{obs} \times K$ matrix of effective row loadings into a compact sparse representation suitable for fast row-wise accumulation in C++. Each row can correspond to a fully moderated loading vector after any item or person covariate effects have already been applied.

Usage

```
rowLoadingsToSparse(loadings, tol = 0)
```

Arguments

loadings	A numeric matrix with one row per observation and one column per latent dimension.
tol	Values with absolute magnitude less than or equal to <code>tol</code> are treated as structural zeros.

Value

A list with:

`row_ptr` Integer vector of length `Nobs + 1`. Entries mark the start of each row in `load_idx` and `load_val`.
`load_idx` 1-based latent-dimension indices for non-zero loadings.
`load_val` Non-zero loading values.
`n_dim` Number of latent dimensions.

<code>scoreContributions</code>	<i>Score contributions of a fitted model</i>
---------------------------------	--

Description

Each response's contribution to the score of the marginal likelihood in its item's parameters, with the matching observed information, and each person's contribution to the score in the ability distribution's mean. They are what score-based tests of differential item functioning and of ability covariates are built from (see [scoreTest\(\)](#)), and they can be summed by any grouping for other diagnostics.

Usage

```
scoreContributions(
  fit,
  nodes = NULL,
  information = TRUE,
  cores = 1L,
  exclude = NULL
)
```

Arguments

<code>fit</code>	A fit from fitIRT() , by either estimator, with <code>normalise = FALSE</code> (or <code>keepInternals = TRUE</code>).
<code>nodes</code>	Integer. Quadrature nodes per dimension. Default: the fit's own rule if it was refined by quadrature, otherwise 9, 7 and 5 for one to three dimensions and the anchored pairwise rule beyond. 1 gives the plug-in score at the posterior mode.
<code>information</code>	Logical. Also return each response's observed information in its item's slots. Default TRUE.
<code>cores</code>	Integer. Threads for the person-level kernels.
<code>exclude</code>	Optional item labels whose responses are held out of every person's posterior – the anchoring step of DIF purification (see <code>purify</code> in scoreTest()). The ability distribution's mean (if the fit estimated it) and the person-covariate coefficients are then re-estimated on the remaining items, the item parameters held, since the excluded items' DIF pulled on them too. Held-out responses are still scored (column held), each against the posterior from the remaining items with its own likelihood added, as if it were the person's only other response.

Details

A person's marginal score is the posterior expectation of the complete-data score (the Fisher identity), taken here by adaptive Gauss-Hermite quadrature centred at the person's Laplace mode; the information is the Louis identity on the same nodes. Under the fitted model these contributions have mean zero for every person, conditionally on the person's covariates, which is the property score-based tests rest on. The plug-in score at an estimated ability does not have it: the ability estimate's own error leaves a residual pattern in ability, and when a covariate is related to ability that pattern is ordered by the covariate. `nodes = 1` returns that plug-in score, at the posterior mode, for comparison.

The scores are those of the quadrature marginal, so they sum to (nearly) zero over an item's responses only when the item parameters sit at that marginal's optimum. A fit that ended on the Laplace objective (the default for a one-dimensional 2PL) is close to it but not at it; `scoreTest()` allows for the gap, but with few responses per person a fit refined with `laplaceRefine = "agq"` is the cleaner basis. Measured with ten items per person, 5,000 persons and a covariate related to ability: refining shrank the item score sums fifteenfold and moved the LM test's false-positive rate from 5.3 to 4.0 per cent (600 tests each).

Scores are in the coordinates the model is estimated in, which are also the ones item covariate effects act on: B is the intercept in $\eta = a'\theta - B$ (positive is harder); A the loading's raw coordinate (softplus link, or the loading itself when signed); C and D the logits of the half-range asymptote links.

Value

An object of class `bigIRT_scores`, a list with

`rows` A data.table, one row per training response: `row` (its row in the data passed to `fitIRT()`), `id`, `Item`, internal indices `person` and `item`, the response `y`, the score in each free parameter type (B, A or A1, A2, ..., C, D; zero where the response's item has that parameter fixed), `eta`, the score in the linear predictor (for the 2PL, the response minus its posterior-expected probability), `ppred`, the probability of a correct response given the person's other responses (the row's leave-one-out posterior predictive), and, with `information`, the observed information entries `I.<a>.<b>`.

`persons` A data.table per person: `id`, `person`, the posterior mean and SD of (base) ability on each dimension, the score in the ability mean (`mu`, `mu1`, ...) and its information `Imu.<k>.<l>`.

`slots`, `K`, `rule`, `item_free`, `item_labels`, `id_name`, `item_name` Bookkeeping used by `scoreTest()`.

See Also

[scoreTest\(\)](#)

scoreTest

Score-based tests of covariate relations after a fit

Description

Tests whether item parameters depend on a covariate (differential item functioning, uniform or not), or whether ability does, using only the fitted model's score contributions (`scoreContributions()`). No model is refitted, so any number of covariates can be screened after one large fit.

Usage

```
scoreTest(
  fit,
  covariates,
  parameters = "B",
  form = c("auto", "linear", "spline", "factor", "fluctuation"),
  by = "item",
  items = NULL,
  df = 3L,
  dims = NULL,
  level = c("auto", "person", "row"),
  ngrid = 50L,
  trim = 0.1,
  nsim = 1000L,
  seed = NULL,
  minN = 20L,
  purify = FALSE,
  purifyAlpha = 0.05,
  purifyMaxIter = 5L,
  information = c("expected", "observed", "opg"),
  adjust = c("auto", "none"),
  adjustDf = 3L,
  adjustMethod = c("fast", "full"),
  effects = TRUE,
  nodes = NULL,
  cores = 1L
)
```

Arguments

<code>fit</code>	A fit from <code>fitIRT()</code> , or its <code>scoreContributions()</code> .
<code>covariates</code>	The covariates to test. Person-level: a data frame with the fit's id column (or id) and one column per covariate. Row-level: a vector or data frame with one row per row of the data passed to <code>fitIRT()</code> (a covariate can then vary within person, e.g. age at the time of the response).
<code>parameters</code>	Which parameters to test: any of "B" (uniform DIF), "A" (every loading of the item), "C", "D", tested jointly; or "Ability".
<code>form</code>	"linear", "spline" (natural cubic spline with df degrees of freedom), "factor", or "fluctuation". Default: "factor" for a factor, character or logical covariate, "linear" otherwise.
<code>by</code>	"item" for one test per item, "all" for one coefficient shared by every item, or a vector naming a group for each item (in the order of <code>fit\$itemPars</code>) for coefficients shared within groups.
<code>items</code>	Optional item labels to test (default all).
<code>df</code>	Degrees of freedom of the spline basis.
<code>dims</code>	For "Ability", the dimensions tested (default all).

level	"auto", "person" or "row"; see covariates.
ngrid, trim, nsim	Fluctuation tests: maximum number of breakpoints (blocks of persons in covariate order), the trimming fraction for maxLM, and the number of simulated null paths (increased tenfold automatically when a p-value lies near the first batch's resolution).
seed	Optional seed for the simulated p-values.
minN	Minimum number of responses for an item to be tested.
purify, purifyAlpha, purifyMaxIter	Iterative purification against DIF contamination (see above), the adjusted-p threshold for flagging, and the maximum number of anchoring rounds. Item tests by item only.
information	Which information the item tests use. "expected" (default): each response's score is exactly the score of the Bernoulli for that response given the person's other responses (with probability ppred), and this is that Bernoulli's Fisher information – the score's variance given the other responses, never negative. On a simulated thin item bank with no DIF (50 to 200 responses an item) it held the nominal rate for intercepts and discriminations alike. "observed": each response's Louis information. A single response's can be negative, and on thin items the observed efficient information of a comparison the item cannot support (a discrimination split by a factor with few responses at one level) can collapse while the scores still vary; such a test is withheld (p = NA) when that information is below a fifth of the outer-product one (info_ratio). "opg": the outer product of the scores, the classical choice for fluctuation tests, but liberal on thin items (7 to 12 per cent at a nominal 5 in the same simulation). The ability tests always use the observed information of the ability mean, which is not subject to these problems.
adjust, adjustDf	"auto" (default) adds a covariate missing from the fit's ability model to it before testing, as a natural spline with adjustDf degrees of freedom (indicators for a factor or a covariate with few values), items held; "none" does not. Needs fit to be a fit; the result records it in ability_adjusted.
adjustMethod	How the added covariate's coefficients and the residual ability SD are estimated. "fast" (default) holds the fit's own coefficients and takes Newton steps for the added ones – a regression of the posterior means on the basis, corrected for their shrinkage – with the residual SD by variance decomposition; a few quadrature passes. It needs a covariate constant within person and a one-dimensional model, and otherwise falls back to "full": every coefficient and the SD re-estimated jointly to convergence, which can take dozens of passes over the data.
effects	Report DIF effect sizes (parametric forms). Each response's score is the score of a Bernoulli given the person's other responses, so the derivative of its predicted probability in the item's parameters is exact; applied to the one-step DIF estimate it gives the response's first-order change in the probability of a correct answer at the person's own ability. These are averaged per item (effects) and, with purify and a person-level covariate, summed per person over the flagged items they answered (dtf).
nodes, cores	Passed to <code>scoreContributions()</code> when fit is a fit.

Details

Two families of test are available.

Parametric (form = "linear", "spline", "factor") The score (Lagrange multiplier) test of adding the covariate, through the chosen basis, to the tested parameters. Exact chi-square reference. Each test also gives the one-step estimate of the coefficients it tests – for items, what `fitIRT()` with the covariate in `BitemPreds` (or `AitemPreds`, ...) and `itemSpecificBetas` = TRUE would estimate, on the same coordinates; for ability, what it would estimate with the covariate in `personPreds` – with a standard error.

Fluctuation (form = "fluctuation") The nonparametric CUSUM tests of Merkle and Zeileis (double maximum DM, Cramer-von Mises CvM, maxLM), which order persons by the covariate and need no functional form. The process's covariance is computed from the information accrued between breakpoints and the p-values are simulated from it, so they stay valid when information does not accrue uniformly along the covariate (impact does that). `p_DM_bb` is the classical Brownian bridge p-value, for comparison.

In both, each item's own other parameters are projected out exactly (Neyman's C(alpha) construction), so testing one parameter while the others are estimated is accounted for, and the small pull of the item priors on the fit is too.

Impact. A covariate related to ability has to be in the ability model for its DIF tests to be valid; otherwise every item inherits the unmodelled ability difference. When `fit` is a fit and the covariate is not among its `personPreds`, `adjust` = "auto" (the default) puts it there after the fact: a natural spline of it (`adjustDf` degrees of freedom), or indicators for a factor, centred, enters the ability regression, its coefficients are estimated with the item parameters held, and the contributions are recomputed under that model before testing. A response-level covariate enters per response, so ability may also change with it within person (age at the response, say). The item parameters stay as the fit estimated them, without the covariate; each item's test projects out the resulting discrepancy in its own parameters. `adjust` = "none" tests against the fit as it is, and then warns when a person-level covariate is related to ability but absent from the model. The same is true of the uniform part of DIF that every item shares: it is indistinguishable from an ability difference, and DIF is measured relative to it.

Purification. When several items have DIF in the same direction, their DIF pulls every person's ability posterior (and the ability model's coefficient on the covariate) towards explaining it, and the clean items then show DIF the other way. `purify` = TRUE screens, flags the items whose Benjamini-Hochberg adjusted p is below `purifyAlpha`, recomputes the score contributions with the flagged items' responses left out of every person's posterior and the ability coefficients re-estimated on the rest (exclude in `scoreContributions()`), and retests the rest against that anchor; flagged items are retested against the same anchor, each response scored against the anchor posterior with its own likelihood added, in the same pass. It repeats until the flagged set stops changing. Only the person side is recomputed, so this needs the fit, not its score contributions, but no refit.

Ability. `parameters` = "Ability" tests the covariate as a predictor of ability (on every dimension, or those in `dims`), with the ability mean and the fit's existing person predictors projected out.

Value

An object of class `bigIRT_scoretest`: a list with `tests`, a `data.table` with one row per covariate and item (or group): the number of responses `n`, the statistic(s), degrees of freedom and p-values (with `purify`, also whether the item ended as an anchor and the stage its test comes from), with

Benjamini-Hochberg (`p_fdr`) and Holm (`p_holm`) adjustments within each covariate; and, for the parametric forms, coefficients, a data.table of one-step estimates (`estimate`), standard errors and z values per tested parameter and basis column; and, when a covariate was added to the ability model (`adjust`), `adjustment`: its basis terms' estimated coefficients, the re-estimated ability SD against the fit's, and the iterations taken and whether they converged (a warning is given if not). With `effects`: `effects`, per item and contrast (a factor level against the reference level; a continuous covariate's 90th against its 10th percentile over responses), the mean change in the probability of a correct answer over the item's responses at equal ability, signed (`dP`) and unsigned (`dP_abs`), first-order in the DIF estimate. They are relative to the scale the other items set: with `purify`, the anchors; without, all items, which absorb any net DIF into the estimated impact. With `purify` and a person-level covariate (one-dimensional models), also `dtf`, the test-level effect of the flagged items per level or covariate quintile, relative to the reference group (the reference level, or the middle quintile): the mean change in a person's expected number correct over the items they answered (`dT_flagged`), and the ability shift that would change their expected score as much (`dtheta_flagged`) – the bias DIF puts into a person's ability estimate when scored with these item parameters, to first order. Without anchors there is no test-level effect to report: with the covariate's ability effect estimated freely, the DIF effects of all items sum to about zero for each group.

See Also

[scoreContributions\(\)](#)

simIRT

Simulate IRT data

Description

Simulate IRT data

Usage

```
simIRT(
  Nsubs = 100,
  Nitems = 200,
  Nscales = 1,
  NitemsAnswered = Nitems,
  ASD = 0,
  AMean = 1,
  BSD = 1,
  BMean = 0,
  logitCSD = 1,
  logitCMean = -2,
  logitDSD = 0,
  logitDMean = 20,
  AbilitySD = 1,
  AbilityMean = 0,
  AbilityCorr = diag(1, Nscales),
```

```

    itemPreds = NA,
    AitemPredEffects = NA,
    BitemPredEffects = NA,
    logitCitemPredEffects = NA,
    logitDitemPredEffects = NA,
    personPreds = NA,
    AbilityPredEffects = NA,
    mirt = FALSE,
    loadingSparsity = 0.3,
    primaryScale = NA,
    loadings = NA,
    crossLoadingSD = 0.15,
    returnRowLoadings = TRUE
  )

```

Arguments

Nsubs	Integer. Number of subjects.
Nitems	Integer. Number of items per scale.
Nscales	Integer. Number of scales.
NitemsAnswered	Integer (or length-Nscales integer vector). Number of items answered per person per scale. Values below Nitems generate sparse-response datasets.
ASD	Numeric. SD of simulated item discrimination values (A)
AMean	Numeric. Mean of simulated item discrimination values (A).
BSD	Numeric. SD of simulated item difficulty values (B).
BMean	Numeric. Mean of simulated item difficulty values (B).
logitCSD	Numeric. SD of simulated guessing values on logit scale.
logitCMean	Numeric. Mean of simulated guessing values on logit scale.
logitDSD	Numeric. SD of simulated upper asymptote values on logit scale.
logitDMean	Numeric. Mean of simulated upper asymptote values on logit scale.
AbilitySD	Numeric. SD of simulated person abilities.
AbilityMean	Numeric. Mean of simulated person abilities.
AbilityCorr	Matrix. Correlation matrix for the latent abilities, used when Nscales > 1. Defaults to the identity.
itemPreds	Optional matrix/data frame of item-level predictors. Rows should align with items.
AitemPredEffects	Optional numeric matrix/vector of effects of itemPreds on item discrimination values.
BitemPredEffects	Optional numeric matrix/vector of effects of itemPreds on item difficulty values.
logitCitemPredEffects	Optional numeric matrix/vector of effects of itemPreds on guessing values (logit scale).

logitDitemPredEffects	Optional numeric matrix/vector of effects of itemPreds on upper asymptote values (logit scale).
personPreds	Optional matrix/data frame of person-level predictors. Rows should align with subjects.
AbilityPredEffects	Optional matrix of effects of personPreds on person ability by scale.
mirt	Logical. If TRUE, simulate a multidimensional response process where each item can load on multiple latent factors. In this mode, Nitems is the total number of items (not items-per-scale).
loadingSparsity	Numeric in $[0, 1]$. Probability that a non-primary loading is active when mirt=TRUE and loadings is not supplied.
primaryScale	Optional integer vector of length Nitems giving each item's primary scale index (1..Nscale) when mirt=TRUE.
loadings	Optional numeric Nitems x Nscale matrix of true item loadings used directly when mirt=TRUE. If omitted, loadings are sampled.
crossLoadingSD	Numeric. SD of sampled non-primary loading values when mirt=TRUE and loadings is not supplied.
returnRowLoadings	Logical. If TRUE and mirt=TRUE, include per-row loading columns (A_1, ..., A_K) in dat.

Value

A list with:

Ability Matrix of true person abilities (Nsubs x Nscale).

A Matrix of true item discriminations (Nitems x Nscale).

B Matrix/vector of true item difficulties. Matrix in legacy mode, vector in mirt=TRUE mode.

The two modes use different parameterisations, and B is not on the same scale as the B that `fitIRT()` reports. Legacy mode generates $p = C + (D - C) \text{logistic}(a(\theta - b))$, so B is a difficulty in the classical sense. mirt = TRUE generates $p = C + (D - C) \text{logistic}(a'\theta - b)$, so B is an intercept, which is the convention `fitIRT()` estimates and reports.

To compare a legacy simulation against a fit, convert first: the generating intercept is $A * B$. Comparing the two unconverted costs an error of $(a - 1)b$, which cancels in mean bias, grows with the size of the difficulty, and does not shrink as the sample grows, so it looks like a calibration failure rather than a units error. `compareIRTmodels()` and `extractMIRTpars()` handle the conversion for you.

C Matrix/vector of true item guessing parameters. Matrix in legacy mode, vector in mirt=TRUE mode.

D Matrix/vector of true item upper asymptote parameters. Matrix in legacy mode, vector in mirt=TRUE mode.

dat Long-format response data as a `data.table`.

Examples

```
sim <- simIRT(Nsubs = 100, Nitems = 40, Nscales = 1, ASD = .2, BSD = .8)
head(sim$dat)
```

testInformation	<i>Item and test information across the ability range</i>
-----------------	---

Description

Fisher information in each response, summed over items, with the conditional standard error of measurement that follows from it.

Usage

```
testInformation(fit, theta = seq(-4, 4, 0.05), scale = NULL, items = NULL)
```

Arguments

fit	A fitted object from <code>fitIRT()</code> .
theta	Numeric vector of ability points.
scale	Optional scale name to restrict to.
items	Optional item index to restrict to.

Value

An object of class `bigIRT_information`, carrying `theta`, `item_information`, `test_information` and `sem`.

tidyIRT	<i>Tidy the item parameters</i>
---------	---------------------------------

Description

One row per item parameter, with a standard error and Wald interval when asked. Standard errors are carried onto the reported parameter scale by the delta method, since discrimination is estimated as a softplus pre-image and the asymptotes as logits; a signed loading (`signedLoadings` in `fitIRT()`) is estimated as itself and needs none. On a fit with signed loadings every free cross-loading also gets a row, term `A_<scale>`, whose standard error comes from the exact covariance (`method = "exact"`). A parameter fixed through `itemDat` is not estimated, so its standard error and interval are NA. Registered against `broom::tidy()` when `broom` is installed.

Usage

```
tidyIRT(
  x,
  conf.int = FALSE,
  conf.level = 0.95,
  method = if (is.null(x$laplaceStatus$certificate$covariance$items)) "profile" else
    "exact",
  cores = 1L,
  ...
)
```

Arguments

<code>x</code>	A fitted object from <code>fitIRT()</code> .
<code>conf.int</code>	Logical. Compute standard errors and intervals.
<code>conf.level</code>	Numeric. Interval coverage.
<code>method</code>	Character. Passed to <code>itemInformation()</code> . By default "exact" when the fit's convergence certificate kept the exact covariance, which costs nothing, and "profile" otherwise.
<code>cores</code>	Integer. Cores for the information calculation.
<code>...</code>	Unused.

Value

A data frame of item, term and estimate and, when requested, standard error and interval.

wleIRT	<i>Compute Weighted Likelihood Estimate (WLE) and Standard Error (SE)</i>
--------	---

Description

Computes the WLE and SE for each subject and scale in a bigIRT model.

Usage

```
wleIRT(fit)
```

Arguments

<code>fit</code>	A fitted object returned by <code>fitIRT()</code> .
------------------	---

Value

A list with:

wle Matrix of weighted likelihood estimates by subject and scale.

wleSE Matrix of approximate standard errors for wle.

Examples

```
sim <- simIRT(Nsubs = 60, Nitems = 20, Nscales = 1)
fit <- fitIRT(sim$dat, pl = 2, cores = 1, dropPerfectScores = FALSE)
wle <- wleIRT(fit)
str(wle)
```

Index

as.mirt, [3](#)
as.mirt(), [5](#)
augmentIRT, [3](#)

checkConvergence, [4](#)
checkConvergence(), [30](#)
checkMirtBridge, [5](#)
checkMirtBridge(), [3](#)
compareIRTmodels, [5](#)
compareIRTmodels(), [49](#)

dropPerfectScores, [6](#)

empiricalICC, [7](#)
empiricalICC(), [29](#)
estfun.bigIRT_scores, [8](#)
extractMIRTpars, [10](#)
extractMIRTpars(), [35](#), [49](#)

fitIRT, [10](#)
fitIRT(), [3–5](#), [8](#), [24–26](#), [28–32](#), [40–44](#), [46](#),
[49–51](#)

glanceIRT, [23](#)

heldoutLogLik, [24](#)
heldoutLogLik(), [25](#), [27](#)
heldoutMetrics, [24](#)
heldoutMetrics(), [31](#)

improveModel, [25](#)
improveModel(), [24](#), [33](#)
IRTcurve, [27](#)
IRTic, [28](#)
itemFit, [28](#)
itemInformation, [20](#), [29](#)
itemInformation(), [41](#), [51](#)

logLik.bigIRT_fit, [30](#)
logLik.bigIRT_fit(), [28](#)
looIRT, [21](#), [31](#)

modificationTests, [32](#)
modificationTests(), [25–27](#)

normaliseIRT, [13](#), [33](#)
normaliseMIRT, [34](#)
normaliseMIRT(), [6](#)

personCovarianceMatrices, [36](#), [38](#)
personCovarianceMatrices_R, [37](#), [38](#)
plotLaplaceDiagnostics, [39](#)

reliability, [21](#), [40](#)
reportIRT, [40](#)
rowLoadingsToSparse, [36](#), [39](#), [41](#)

sandwich::estfun(), [8](#)
scoreContributions, [42](#)
scoreContributions(), [8](#), [24](#), [43–47](#)
scoreTest, [43](#)
scoreTest(), [9](#), [33](#), [42](#), [43](#)
simIRT, [47](#)

testInformation, [50](#)
tidyIRT, [50](#)

wleIRT, [51](#)