Package: birtRcpp (via r-universe)

October 31, 2026

Type Package

Title BIvariate Response Time Modeling with 'Rcpp'

Version 0.3.0

Description Item response and response time models fitted from their full conditionals. A built-in model is chosen with ecm(data, model = ...) or gibbs(data, model = ...): the two-parameter logistic model, alone or with log-normal response times (van der Linden, 2007, <doi:10.1007/s11336-006-1478-z>), latent regressions of ability and speed, cross-relations between ability and the log response times, and their quantile (asymmetric Laplace) versions. The main estimator is ECM (ecm(); Meng and Rubin, 1993, <doi:10.1093/biomet/80.2.267>): one conditional maximization per block of parameters, Polya-Gamma bounds for logistic likelihoods (Polson, Scott and Windle, 2013, <doi:10.1080/01621459.2013.829001>), adaptive or Bock-Aitkin quadrature (Bock and Aitkin, 1981, <doi:10.1007/BF02293801>) with normal-linear latent variables integrated exactly, and Louis' standard errors; quantile models maximize the asymmetric Laplace working likelihood, with sandwich standard errors. The same steps drive Gibbs sampling (gibbs(); conjugate updates and Metropolis-adjusted Langevin steps), with samplers in C++ ('RcppArmadillo') that port the 'Julia' package 'ExtendedRtIrtModeling.jl' (its function names are kept as aliases). Both estimators have a check: check_ecm(), and check_sampler() with the Geweke (2004, <doi:10.1198/016214504000001132>) test and simulation-based calibration. equations() and algorithm() write out the model and every step, before or after fitting.

License GPL-3

URL https://github.com/jiewenTsai/birtRcpp, https://github.com/jiewenTsai/birtExamples

BugReports https://github.com/jiewenTsai/birtRcpp/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports Rcpp (>= 1.0.10), coda, dplyr, purrr, stats, statmod, tibble, utils, graphics, grDevices

LinkingTo Rcpp, RcppArmadillo

Suggests testthat (>= 3.1.7), withr, knitr, rmarkdown, loo, strucchange, sandwich, posterior, LNIRT

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

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

Date/Publication 2026-10-29 10:35:20 UTC

RemoteUrl https://github.com/jiewenTsai/birtRcpp

RemoteRef HEAD

RemoteSha fa2e3dc270d1bc47d98979efcbfcb071f010918a

Contents

birtRcpp-package . . . 3
algorithm . . . 4
algorithm.block_model_vi . . . 6
anova.rtirt . . . 9
block_model . . . 10
block_steps . . . 11
blocks . . . 12
check_ecm . . . 16
check_sampler . . . 17
check_vi . . . 20
ci_test . . . 22
compare_para . . . 23
compose . . . 23
convergence . . . 26
dic . . . 27
draw_shift . . . 28
ecm . . . 29
ecm.block_model . . . 31
ecm_control . . . 36
em_components . . . 37
equations . . . 40
estfun.rtirt . . . 41
estimates . . . 42
fit_indices . . . 44
get_bias . . . 44
gibbs.block_model . . . 45

gibbs.input_data . . . 46
input_data . . . 48
input_para . . . 49
julia_aliases . . . 50
mala . . . 52
marginal_loglik . . . 54
mml . . . 54
models . . . 56
normal_parts . . . 58
person_fit . . . 60
plot.rtirt . . . 61
preknowledge_test . . . 62
reliability . . . 63
rtirt_priors . . . 63
rtirt_qset . . . 65
rtirt_summary . . . 66
rtirt_tbl . . . 67
run_em . . . 68
run_sampler . . . 70
run_simulation . . . 72
score_test . . . 73
score_test.block_model_ecm . . . 74
scores . . . 77
set_cond . . . 78
simulation . . . 79
summary.rtirt . . . 79
vi . . . 80

Index **81**

`birtRcpp-package` *birtRcpp: BIvariate Response Time Modeling with 'Rcpp'*

Description

Built-in models of response accuracy and response times (ports of ExtendedRtIrtModeling.jl), named by `model` and fitted with `ecm()` (the main estimator) or `gibbs()`; `algorithm()` and `equations()` show the steps and the model, `check_ecm()` and `check_sampler()` check them:

```
dat <- input_data(resp = items, time = times, cov = covs, id = "IDSTUD", data = raw)
fit_ecm <- ecm(dat, model = "cross")
fit <- gibbs(dat, model = "cross", init = fit_ecm, n_iter = 10000, n_chain = 3, seed = 1)
summary(fit)
estimates(fit)
scores(fit)
```

Details

To start, these are all you need (the other help pages are for diagnostics, simulation, the building blocks of the samplers, the Julia names and deprecated functions):

step	function
data	`input_data()`
fit	`ecm()` (seconds; estimates, standard errors, AIC, BIC), `gibbs()` (the posterior)
results	`summary()`, `estimates()`, `scores()`, `reliability()`
compare models	`anova()` (`anova.rtirt()`), `fit_indices()`
check a Gibbs fit	`convergence()`, `plot(fit, "trace")` (`plot.rtirt()`)
settings	`ecm_control()`, `gibbs_control()`, `rtirt_priors()`
write up	`equations()` (the model, also in LaTeX)

The tutorial (Traditional Chinese) explains every line: https://github.com/jiewenTsai/birtExamples/blob/main/birtRcpp/tutorial/birtRcpp_intro.md.

The declarative `block_model()`, `vi()` and `mml()` are deprecated (they still work). The accessors have the names of the birt package. Julia names are aliases (julia_aliases). The generics shared with birt (`estimates`, `scores`, `reliability`, `fit_indices`, `equations`, `convergence`, `score_test`, `algorithm`) dispatch to the methods of both packages whichever was attached last: when both are loaded, each package's methods are registered in the other's generics.

Author(s)

Maintainer: Jie-Wen Tsai <tsai.jiewen@gmail.com>

See Also

Useful links:

- https://github.com/jiewenTsai/birtRcpp
- https://github.com/jiewenTsai/birtExamples
- Report bugs at https://github.com/jiewenTsai/birtRcpp/issues

---

`algorithm` *Computation steps of a fit*

Description

Lists the steps of the algorithm behind a fit: for `gibbs()` every full conditional distribution in the order of one iteration (with `collapse = TRUE` the partially collapsed version), for `ecm()` the E- and M-steps. The prior constants of the fit are substituted into the formulas, and each step names the building block that draws it (blocks), so a step can be run, replaced or checked from R. Works before estimation too: give the data and the model as in `ecm()` and `gibbs()`.

Usage

```
algorithm(object, ...)

## S3 method for class 'rtirt_qset'
algorithm(
  object,
  method = NULL,
  collapse = NULL,
  format = c("unicode", "latex"),
  detail = TRUE,
  ...
)

## S3 method for class 'rtirt'
algorithm(
  object,
  method = NULL,
  collapse = NULL,
  format = c("unicode", "latex"),
  detail = TRUE,
  ...
)

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

## S3 method for class 'input_data'
algorithm(
  object,
  model = NULL,
  itemtype = "2pl",
  speed_var = c("free", "fixed"),
  quantile = NULL,
  cov = NULL,
  intercept = FALSE,
  method = NULL,
  collapse = NULL,
  format = c("unicode", "latex"),
  detail = TRUE,
  ...
)
```

Arguments

`object`	A fit of `ecm()` or `gibbs()`, data from `input_data()` (with `model`), or a model object of the deprecated constructors (models) or an `rtirt_qset`.
`...`	Unused.

`method`	`"gibbs"` or `"ecm"`; by default the method of the fit (`"gibbs"` for an unfitted model).
`collapse`	For `"gibbs"`: the partially collapsed sampler; by default the setting of the fit (`FALSE` for an unfitted model).
`format`	`"unicode"` or `"latex"` (an `aligned` block).
`detail`	`TRUE` (default): every step with its formulas; `FALSE`: only the titles of the steps, an overview of one iteration.
`x`	An `rtirt_algorithm` object.
`model`, `itemtype`, `speed_var`, `quantile`, `cov`, `intercept`	For `input_data`: the model as in `ecm()` and `gibbs()`, e.g. `algorithm(dat, model = "cross", method = "ecm")`.

Value

An `rtirt_algorithm` object (printed); `as.character()` gives the lines.

See Also

`equations()` for the model, blocks for the steps as R functions.

Examples

```
cond <- set_cond(n_subj = 100, n_item = 5)
dat <- sim_data(cond, sim_para(cond, "cross"), "cross")
algorithm(dat, model = "cross") # Gibbs full conditionals
algorithm(dat, model = "cross", collapse = TRUE) # partially collapsed Gibbs
algorithm(dat, model = "cross", method = "ecm") # ECM steps
algorithm(dat, model = "cross", format = "latex")
algorithm(dat, model = "cross", detail = FALSE) # the steps only
```

---

algorithm.block_model_vi

Variational Bayes (CAVI) for a declared model

Description

Fits a `block_model()` by coordinate ascent variational inference with the same steps as `gibbs()` and `ecm()`. The three engines use the same full conditionals:

Usage

```
## S3 method for class 'block_model_vi'
algorithm(object, format = c("unicode", "latex"), ...)

## S3 method for class 'block_model'
vi(
  model,
```

```
  steps = NULL,
  latent = NULL,
  nodes = NULL,
  init = NULL,
  cycles = 6,
  max_iter = 1000,
  tol = 1e-09,
  verbose = FALSE,
  quadrature = c("adaptive", "ba81"),
  ...
)

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

Arguments

`format`	`"unicode"` (default) or `"latex"`.
`...`	Unused.
`model`	A `block_model()`.
`steps`	Steps, as for `gibbs()` (the same list can be used for both); by default Gibbs (conditional maximization) where possible and MALA (gradient) otherwise.
`latent`	Parameters integrated out in the E-step (one to three `[person]` parameters with a normal prior).
`nodes`	Quadrature points per latent variable (default 21, 11 or 7 for one, two or three); one number, or one per latent variable (named, e.g. `c(theta = 11, zeta = 5)`).
`init`	Optional starting values (named list).
`cycles`	Maximum number of recentring rounds of the quadrature.
`max_iter`, `tol`	Passed to `run_em()`.
`verbose`	Print the ELBO every 25 sweeps.
`quadrature`	`"adaptive"` (default): q(person) on nodes centred and scaled per person. `"ba81"`: the same nodes for every person (Bock & Aitkin, 1981; default 41, 21 or 11 per latent variable), with the logistic parts summed over persons at each node, as in `ecm()`; faster for many persons (the ELBO is that of the common nodes; use the adaptive nodes when a latent variable has a narrow posterior).
`x`, `object`	A fit of `vi()`.

Details

step	Gibbs (`gibbs()`)	ECM (`ecm()`)	CAVI (`vi()`)
logistic part	draw $\omega \sim PG(1, \eta)$	$E[\omega]$ at η	$q(\omega) = PG(1, c)$, $c^2 = E_q[\eta^2]$
person parameters	draw	integrate (adaptive quadrature)	free-form $q(\theta_j)$ on the same grid
`step_gibbs(x)`, normal prior	draw $N(m/p, 1/p)$	move to m/p	$q(x) = N(Em/Ep, 1/Ep)$ (truncat
`step_gibbs(v)`, variance	draw IG	mode	$q(v)$ = IG with expected squared res

Expectations of products such as $E_q[(a(\theta - b))^2]$ are exact: the expressions are multilinear, and under the factorized q every product is a product of first and second moments. The objective is the evidence lower bound (ELBO); every update raises it. A latent variable integrated exactly (normal and linear, e.g. speed) uses the steps of `ecm()` when its parts are affine in all latent variables: q(u | grid) from a quadratic per person, and in the updates each person's mean and covariance of the latent variables through 2L points (exact for these quadratic parts; Julier & Uhlmann, 2004).

Known behavior of mean-field VB, also here: posterior means are close to the posterior, but posterior SDs of the item parameters are too small (the correlations between parameters are ignored); use `gibbs()` for intervals. `step_mala()` has no CAVI update (the parameter needs a conditionally conjugate full conditional) and `step_shift()` is skipped.

Value

A list of class `block_model_vi`: `summary` (posterior means and SDs of q), `latent` (posterior means and SDs of the person parameters), `q`, `elbo` (trace, one value per sweep), `elbo_drop` (the largest relative drop between sweeps of one round: 0 when every update is right), `iterations`, `converged`, `time`, `model`, `steps`; `algorithm(fit)` lists the updates.

References

Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association, 112*, 859–877.

Bock, R. D., & Aitkin, M. (1981). Marginal maximum likelihood estimation of item parameters: Application of an EM algorithm. *Psychometrika, 46*(4), 443–459. doi:10.1007/BF02293801

Durante, D., & Rigon, T. (2019). Conditionally conjugate mean-field variational Bayes for logistic models. *Statistical Science, 34*, 472–485.

Julier, S. J., & Uhlmann, J. K. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE, 92*(3), 401–422. doi:10.1109/JPROC.2003.823141

Examples

```
set.seed(1); N <- 300; K <- 6
th <- rnorm(N); a0 <- runif(K, 0.8, 2); b0 <- seq(-1.5, 1.5, length.out = K)
Y <- matrix(rbinom(N * K, 1, plogis(sweep(outer(th, a0), 2, a0 * b0))), N)
m <- block_model(Y = Y,
  theta[person] ~ normal(0, 1),
  a[item] ~ normal(1, 1, lower = 0),
  b[item] ~ normal(0, 3),
  Y ~ bernoulli_logit(a * (theta - b)))
fit <- vi(m, list(step_gibbs(theta, a, b)))
fit
algorithm(fit)
```

anova.rtirt *Compare fits of the same data*

Description

anova(fit0, fit1, ...) compares ecm() fits of the same responses and response times, as anova() of lm(): the number of parameters, log-likelihood, AIC and BIC of each fit, and the likelihood ratio test of each fit against the one in the row above (Chisq = 2 * (logLik - logLik above), Df the difference in the number of parameters). The rows are ordered by the number of parameters. The test assumes that each fit is nested in the next, e.g. "null" in "latreg" (the regression coefficients 0) or in "cross" (equal rho), itemtype = "1pl" in "2pl", speed_var = "fixed" in "free"; AIC and BIC also compare fits that are not nested. A fit with more parameters but a lower log-likelihood gives a warning (not nested, or not converged).

Usage

```
## S3 method for class 'rtirt'
anova(object, ...)

## S3 method for class 'rtirt_anova'
print(x, digits = 3, signif.stars = getOption("show.signif.stars", TRUE), ...)
```

Arguments

object, ...	Fits of ecm() (or of gibbs()) of the same data.
x	The result of anova().
digits	Decimals (log-likelihood, AIC, BIC, Chisq, p-value; npar and Df are counts).
signif.stars	Significance stars after the p-value, as in stats::printCoefmat().

Details

For gibbs() fits the table has the criteria of fit_indices() (DIC, WAIC, LOOIC with its standard error), smaller is better, and no test. Fits with prior = "map" (no likelihood), quantile fits (an ALD working likelihood) and a mix of ECM and Gibbs fits are refused.

Value

A data frame of class rtirt_anova (and anova), printed with a heading that names the fits.

Examples

```
cond <- set_cond(n_subj = 300, n_item = 8, n_feat = 1)
dat <- sim_data(cond, sim_para(cond, "latreg"), "latreg")
fit0 <- ecm(dat, model = "null")
fit1 <- ecm(dat, model = "latreg")
anova(fit0, fit1)
```

`block_model` *Declare a model for the building blocks (deprecated)*

Description

Deprecated: `block_model()` and its steps still work (with a warning once per session) but get no new features. Use the built-in models, `ecm(data, model = ...)` and `gibbs(data, model = ...)`.

Usage

```
block_model(...)

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

## S3 method for class 'block_model'
equations(object, format = c("unicode", "latex"), ...)
```

Arguments

`...`	Data (named arguments) and declarations (formulas).
`object`, `x`	A declared model.
`format`	`"unicode"` (default) or `"latex"` (an `aligned` block, as `equations()` for the built-in models).

Details

`block_model()` declares a model once, the way PyMC does: the data, and one line `name ~ distribution(...)` per parameter and per observed matrix. Samplers are then stacked as steps that only name the parameters they update (`step_gibbs()`, `step_mala()`, `step_shift()`, run by `gibbs()`). Each step derives what it needs from the model, so

- the steps always target the same posterior (a prior cannot be forgotten in one step);
- the Pólya–Gamma variables ω of a logistic part are hidden: they are drawn whenever a Gibbs step needs them and are fresh (drawn once per iteration and again after a MALA step has moved a parameter of that part);
- positive parameters are moved on the log scale by MALA, with the Jacobian added;
- dimensions are declared once and checked.

Data: named arguments. Matrices are persons x items (all the same size); vectors have one value per person (length N) or per item (length K); missing responses or log times are `NA`.

Parameters: `name[person] ~ ...`, `name[item] ~ ...` or `name ~ ...` (a single value).

Distributions

- `normal(mean, sd, lower = -Inf, upper = Inf)`: `mean` and `sd` may use parameters and data, e.g. `zeta[person] ~ normal(beta * theta, sqrt(tau2))`. For an observed matrix the mean is written with the person and item parameters as they are; they are expanded to the matrix.

- `bernoulli_logit(eta)`: logistic responses, e.g. `Y ~ bernoulli_logit(a * (theta - b))`.
- `half_t(df, scale)` and `inv_gamma(shape, scale)`: priors of a **variance** v that is used as sd = `sqrt(v)`; `half_t()` is the prior of the SD $\sqrt{v}$ (drawn through the inverse-gamma auxiliary variable of Huang & Wand, 2013).

Expressions may use `+ - * / ^`, `exp()`, `log()` and `sqrt()`. A line `name <- expression` defines a name for later lines, as in `rethinking::ulam()`, e.g. `eta <- a * (theta - b)` and then `Y ~ bernoulli_logit(eta)`; definitions are substituted (they are not parameters).

Value

An object of class `block_model`; `equations()` prints the model.

See Also

`gibbs()` and the steps; `run_sampler()` for samplers written by hand.

Examples

```
set.seed(1); N <- 300; K <- 6
th <- rnorm(N); a0 <- runif(K, 0.8, 2); b0 <- seq(-1.5, 1.5, length.out = K)
Y <- matrix(rbinom(N * K, 1, plogis(sweep(outer(th, a0), 2, a0 * b0))), N)
m <- block_model(Y = Y,
  theta[person] ~ normal(0, 1),
  a[item] ~ normal(1, 1, lower = 0),
  b[item] ~ normal(0, 3),
  Y ~ bernoulli_logit(a * (theta - b)))
m
fit <- gibbs(m, list(step_mala(theta), step_gibbs(a, b)), n_iter = 600)
summary(fit)[1:4, ]
algorithm(fit)
```

---

block_steps *Steps for declared models*

Description

Steps name the parameters they update; `gibbs()` runs them in the order given, once per iteration. What a step does is derived from the `block_model()`:

Usage

```
step_gibbs(..., cut = character())

step_mala(..., step = 0.1)

step_shift(...)
```

Arguments

...	Parameter names (unquoted); for step_shift() name = direction.
cut	Names of observed data whose parts are left out of this Gibbs step (a cut).
step	Starting step size of MALA.

Details

- step_gibbs(...): draws each parameter from its full conditional, one after the other. A parameter with a normal prior gets its normal full conditional: the precision and linear term are added up over every part of the model that uses it (normal parts directly; a logistic part through its Pólya–Gamma variables, which are drawn automatically when needed). A variance with a half_t() or inv_gamma() prior gets its conjugate update. The model must be linear in the parameter (otherwise use step_mala()). With cut = "logT" the parts of the data named there do not inform this step (a cut; Plummer, 2015): e.g. step_gibbs(theta, cut = "logT") draws the ability from the responses only, while the response-time parameters still use the ability. The chain then targets the cut distribution, not the posterior (so the Geweke check of check_sampler() does not apply), and ecm() and vi() refuse a cut.
- step_mala(..., step = 0.1): one MALA step (Roberts & Tweedie, 1996) per person or item, for the named parameters jointly (they must have the same dimension); the logistic parts use their exact likelihood (no Pólya–Gamma variables). A parameter with a lower bound is moved on the log scale with the Jacobian; the step sizes adapt during burn-in.
- step_shift(...): moves several parameters together along a direction, e.g. step_shift(theta = 1, b = 1) (theta + c and b + c) or step_shift(rho = -1, zeta = theta), with c drawn from its exact conditional (a generalized Gibbs step; Liu & Sabatti, 2000). The logistic parts must not change along the direction, and the normal parts must be linear in c. It speeds up directions that only the prior identifies.

Value

A step (a list) for gibbs().

---

blocks *Building blocks for writing samplers in R*

Description

Each draw_*() function draws one block from its full conditional and is named after the C++ function of the samplers that it calls (irt_part() draws nothing: it returns the accuracy part of the conditional of ability). algorithm() names the function of every step of gibbs().

Usage

```
draw_gauss(prec, num)

draw_invgamma(shape, scale)
```

```
draw_tnorm(mean, sd, lower = -Inf, upper = Inf)

draw_omega(a, b, theta)

draw_probit_z(Y, eta)

draw_pg(eta)

irt_part(Y, omega, a, b)

draw_items_pg(
  Y,
  omega,
  theta,
  a,
  priors = rtirt_priors(),
  one_pl = FALSE,
  b_mean = NULL
)

draw_items_joint(Y, theta, a, b, priors = rtirt_priors(), b_mean = NULL)

draw_lambda(L, m, k1nu, W, mu, sd, lower = 0)

draw_sigma2(
  L,
  m,
  lambda,
  current,
  priors = rtirt_priors(),
  nu = NULL,
  q = NULL
)

draw_var(alpha, beta, current, prior = c(df = 3, scale = 1))

draw_nu(r, s, q)

draw_theta_collapsed(prec, num, W, y, rho, m0, g, tau)

draw_rt_items_collapsed(
  L,
  k1nu,
  W,
  theta,
  mz,
  tau,
  with_rho,
```

```
  mu,
  sd,
  rho_sd,
  lambda,
  rho,
  lower = 0
)
```

Arguments

`prec`, `num`, `shape`, `scale`, `mean`, `sd`, `lower`, `upper`	Numeric vectors (see Details).
`a`, `b`, `theta`, `lambda`, `rho`, `mz`, `m0`, `g`, `tau`	Numeric vectors.
`Y`, `L`, `omega`, `m`, `k1nu`, `W`, `y`, `nu`, `r`, `s`, `eta`	Numeric matrices (persons x items).
`priors`	Priors from `rtirt_priors()`.
`one_pl`	`TRUE`: keep a fixed.
`b_mean`	Prior means of b, one per item (`NULL`: the mean of `priors$b`).
`mu`	Prior mean (and `sd` the prior SD) of the time intensities; `lower` is their lower truncation.
`current`	Current values (variances).
`q`	Quantile level of the ALD.
`alpha`, `beta`	Exponents of the likelihood of a variance.
`prior`	Prior of a variance: `c(df, scale)` or `c(shape, scale)`.
`with_rho`	Draw rho with lambda.
`rho_sd`	Prior SD of rho.

Details

The full conditionals used by `gibbs()`, exposed so that new samplers (other blockings, cut or semi-modular schemes, nested chains) can be put together in R while every draw still runs in C++. All of them use R's random number generator, so `set.seed()` makes a composed sampler reproducible, and a sampler written with them reproduces `gibbs()` step by step. Notation as in the samplers: persons j = 1..N, items i = 1..K, `Y` the 0/1 responses, `L` the log response times, logit P(y = 1) = a (theta - b), log T = lambda - zeta - rho theta + e.

- `draw_gauss(prec, num)`: independent draws from N(num / prec, 1 / prec), elementwise (any conditionally normal scalar: ability, speed, rho_i, a shift).
- `draw_invgamma(shape, scale)`, `draw_tnorm(mean, sd, lower, upper)`: inverse gamma and truncated normal draws, elementwise.
- `draw_omega(a, b, theta)`: Pólya–Gamma variables PG(1, a_i (theta_j - b_i)), an N x K matrix.
- `draw_pg(eta)`: Pólya–Gamma variables PG(1, eta) for any matrix of linear predictors (models whose logit is not a (theta - b), e.g. several abilities or covariates in the item response).

- `draw_probit_z(Y, eta)`: for a probit response model, the latent normal responses of Albert & Chib (1993), N(eta, 1) truncated to the positive (y = 1) or negative (y = 0) half line; missing y (NA) are not truncated. Given them, the probit model is a normal linear model.
- `irt_part(Y, omega, a, b)`: the accuracy part of the conditional of ability given omega, a normal part (precision `prec` and linear term `num`, see normal_parts): `draw(irt_part(Y, omega, a, b) + normal_prior(0, 1))`.
- `draw_items_pg(Y, omega, theta, a, priors, one_pl, b_mean)`: b, then a, given omega (Gibbs); returns `list(a, b)`.
- `draw_items_joint(Y, theta, a, b, priors, b_mean)`: (a, b) jointly with omega integrated out (Metropolis–Hastings with a Fisher-scoring proposal in (a, -a b)); returns `list(a, b)` with the attribute `accepted` (one logical per item). It mixes much better than `draw_items_pg()` for very easy or very difficult items; draw omega anew after it.

The item priors are those of `rtirt_priors()` (`a`: truncated normal on a > 0, `b`: normal); `b_mean` gives b a prior mean per item (e.g. `W %*% delta` in an LLTM with item covariates `W`), in place of the mean of `priors$b`.

- `draw_lambda(L, m, k1nu, W, mu, sd, lower)`: time intensities given the RT means `m` (N x K, e.g. -zeta - rho theta), the ALD shifts `k1nu` (zeros for normal times) and the precisions `W` (N x K); prior N(mu, sd^2) on (lower, Inf).
- `draw_sigma2(L, m, lambda, current, priors, nu = NULL, q = NULL)`: residual variances (normal) or ALD scales (with the mixing variables `nu` and the quantile `q`), from their `current` values (a half-t prior draws its inverse-gamma auxiliary variable from them).
- `draw_var(alpha, beta, current, prior)`: a variance with likelihood var^-alpha exp(-beta / var) (n residuals u ~ N(0, var): `alpha = n / 2`, `beta = sum(u^2) / 2`) and the prior `c(df, scale)` (half-t on its square root, two inverse-gamma steps) or `c(shape, scale)` (inverse gamma).
- `draw_nu(r, s, q)`: ALD mixing variables for residuals `r` and scales `s` (matrices).
- `draw_theta_collapsed(prec, num, W, y, rho, m0, g, tau)` and `draw_rt_items_collapsed(L, k1nu, W, theta, mz, tau, with_rho, mu, sd, rho_sd, lambda, rho)`: the collapsed steps of `gibbs(collapse = TRUE)` (speed integrated out).

Missing responses (NA in `Y`) are skipped by `irt_part()`, `draw_items_pg()` and `draw_items_joint()`. `draw_lambda()` and `draw_sigma2()` need complete log times (they stop on NA); with missing times use `calc_normal_part()` with `draw()` and `draw_var()`. `run_sampler()` runs a sampler written from these blocks (chains, burn-in, storage, summary) and `check_sampler()` checks it.

These are low-level: arguments are checked only for their dimensions, and the caller is responsible for a valid sampler (each block must be drawn from its conditional given the current values of everything else, or from a marginal followed by the integrated variable). `inst/examples/ex07_blocks.R` writes the `rtirt_cross()` sampler with them; `ex08_testlet.R`, `ex09_mirt_rt.R` and `ex10_conditional_dependence.R` compose models from the literature.

Value

Draws (vectors, matrices or lists, see Details).

Examples

```
set.seed(1)
draw_gauss(prec = c(1, 4), num = c(0, 2))          # N(0, 1) and N(0.5, 1/4)
theta <- rnorm(200); Y <- matrix(rbinom(600, 1, plogis(theta)), 200)
om <- draw_omega(a = rep(1, 3), b = rep(0, 3), theta = theta)
str(draw_items_pg(Y, om, theta, a = rep(1, 3), priors = rtirt_priors()))
```

---

check_ecm *Check an EM algorithm*

Description

The EM counterpart of the Geweke check of `check_sampler()`. A Gibbs sampler is right when every step leaves the posterior unchanged; an EM algorithm is right when

Usage

```
check_ecm(
  model = NULL,
  steps = NULL,
  latent = NULL,
  nodes = NULL,
  prior = c("map", "ml"),
  n_start = 10,
  data = NULL,
  init = NULL,
  step = NULL,
  loglik = NULL,
  positive = character(),
  seed = NULL,
  quadrature = c("adaptive", "ba81")
)

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

Arguments

`model`	An `ecm()` fit (e.g. `ecm(dat, model = "cross")`), a `block_model()`, or `NULL` for a hand-written EM.
`steps`, `latent`, `nodes`, `prior`, `quadrature`	As in `ecm()` (for a built-in model only `prior` and `nodes`, default 41).
`n_start`	Number of starting values.
`data`, `init`, `step`, `loglik`, `positive`	A hand-written EM, as in `run_em()`.
`seed`	Random seed.
`x`	An `em_check` object.
`...`	Unused.

Details

1. **every step never lowers the objective** (from many starting values, not only from the path that EM happened to take);
2. **the E-step agrees with the objective** (Fisher's identity: the gradient of the log marginal likelihood equals the gradient of the expected complete-data log likelihood $Q(\cdot \mid x)$ at x); and
3. **ECM stops at a maximum** (a general optimizer started at the ECM estimate finds nothing better, and the gradient is zero there).

For an `ecm()` fit of a built-in model (a `gibbs()` fit or a model object of the deprecated constructors also works) the three are checked for the EM of `ecm()` with its own E- and M-step, from `n_start` starting values jittered around those of `ecm()`: one whole M-step (which updates all blocks) never lowers the objective, the analytic score (Fisher's identity) equals the numerical gradient of the objective, and the fit is a maximum. For a quantile model (one level, e.g. `check_ecm(ecm(dat, model = "cross", quantile = 0.9))`) the objective is the ALD working log-likelihood, the starts are jittered around the fit, and `prior` is not used. For an EM written by hand, give `data`, `init`, `step` and `loglik` as for `run_em()`; checks 1 (from `n_start` jittered starts) and 3 are done. For a `block_model()` (deprecated) all three are checked, with starting values drawn from the declared priors.

Value

A data frame with one row per check (`check`, `worst`, `limit`, `passed`), class `em_check`.

Examples

```
# the EM of ecm() for the cross-relation model
cond <- set_cond(n_subj = 200, n_item = 5)
dat <- sim_data(cond, sim_para(cond, "cross"), "cross")
fit <- ecm(dat, model = "cross", se = FALSE)
check_ecm(fit, prior = "ml", n_start = 3, seed = 1)
# an EM written by hand (the 2PL of run_em())
set.seed(1)
Y <- matrix(rbinom(300 * 5, 1, plogis(outer(rnorm(300), seq(-1, 1, length.out = 5), "-"))), 300)
step <- function(s, d) {
  e <- estep_irt(d$Y, s$a, s$b)                # E-step
  s[c("a", "b")] <- mstep_items(e, s$a, s$b)    # M-step
  s$.loglik <- e$loglik
  s
}
loglik <- function(s, d) estep_irt(d$Y, s$a, s$b)$loglik
check_ecm(data = list(Y = Y), init = list(a = rep(1, 5), b = rep(0, 5)), step = step,
          loglik = loglik, positive = "a", n_start = 3, seed = 1)
```

`check_sampler` *Check a Gibbs sampler by simulation*

Description

Tests whether the sampler of `gibbs()` for a model draws from the correct posterior, with two general methods that need only the prior and the likelihood:

Usage

```
check_sampler(
  model = NULL,
  method = c("geweke", "sbc"),
  n_persons = 20,
  n_iter = NULL,
  n_rep = 200,
  n_draws = 99,
  collapse = NULL,
  itemtype = NULL,
  priors = rtirt_priors(),
  sampler_priors = priors,
  probs = c(0.1, 0.25, 0.5, 0.75, 0.9),
  n_prior = 1e+05,
  seed = NULL,
  steps = NULL,
  monitor = NULL,
  step = NULL,
  prior = NULL,
  simulate = NULL,
  ...
)

## S3 method for class 'rtirt_check'
print(x, n = 10, ...)
```

Arguments

`model`	An `ecm()` or `gibbs()` fit (e.g. `ecm(dat, model = "cross", se = FALSE)`; only its model, settings and size are used), or a model object of the deprecated constructors. Or a `block_model()` (deprecated): its stacked steps (`steps`, see `gibbs()`) are checked by the Geweke test, with the parameters drawn from the declared priors and the data from the declared model (use a small data set, e.g. 20 persons and 4 items). `NULL` with `step`.
`method`	`"geweke"` or `"sbc"`.
`n_persons`	Persons per simulated data set.
`n_iter`	Geweke: iterations of the successive-conditional chain (the first 10% are dropped; default 2e5, about 15 s for three items). SBC: iterations per fit (the first half is burn-in; default 1e4).
`n_rep`	SBC: number of simulated data sets.
`n_draws`	SBC: thinned posterior draws per fit (ranks 0 to `n_draws`).

`collapse`, `itemtype`
: Sampler settings, as in `gibbs()` (default: those of the model).

`priors`
: Priors of the simulation and the sampler, from `rtirt_priors()`.

`sampler_priors`
: Priors given to the sampler (default `priors`).

`probs`
: Geweke: probabilities of the prior quantiles that are compared.

`n_prior`
: Geweke: prior draws for the reference distribution.

`seed`
: Random seed.

`steps`
: For a `block_model()`: the steps (default as in `gibbs()`).

`monitor`
: For a `block_model()`: the parameters compared (default: all but the person parameters); with `step`: the names of the state elements compared (required).

`step`, `prior`, `simulate`
: A sampler written by the user: `step(s, data)` returns the new state, `prior()` a state drawn from the prior, `simulate(s)` the data list given the state.

`...`
: Unused.

`x`
: An `rtirt_check` object.

`n`
: Rows printed.

Details

- `method = "geweke"` (Geweke, 2004): the successive-conditional simulator. One sweep of the sampler given the data alternates with new person values and a new data set simulated given the current parameters. (Simulating new data from the old person values is also exact, but the persons are then held by their own data and the chain crawls: a false alarm in practice.) If the sampler is correct, the chain of the parameters has the **prior** as its distribution. For every parameter and each probability in `probs`, the share of the chain below the prior quantile is compared with the probability (z-test with the effective sample size of the chain); p-values are Holm-adjusted over all comparisons. Cheap: one sweep per iteration on a small data set.
- `method = "sbc"`, simulation-based calibration (Cook, Gelman & Rubin, 2006; Talts et al., 2018; Modrák et al., 2025): `n_rep` times, draw the parameters from the prior, simulate data, run the sampler and record the rank of the true value among `n_draws` thinned posterior draws. The ranks are uniform for a correct sampler; a chi-square test over 10 bins per parameter (Holm-adjusted).

Both use the model only as a template: its type, quantile, settings, number of items and (the first `n_persons` rows of) its covariates. Data are simulated, so a small `n_persons` keeps the posterior wide and the check fast. The priors must be proper and must not depend on the data (`rtirt_priors_julia()` centres lambda on the data and cannot be used).

`sampler_priors` lets the sampler use other priors than the simulation: a check of the check, which must then report a failure.

Chain length matters for both. The successive-conditional chain must travel over the whole prior, and its parameters move by about one posterior width per iteration, so parameters with wide priors (lambda ~ N(0, 10^2)) need long chains; the print method warns when an effective sample size is small. In SBC the posterior draws of each fit must be close to independent after thinning: with short fits a slowly mixing parameter (such as rho) piles its ranks up at one end, which looks like an error of the sampler. Both effects disappear with longer chains, a real error does not. **Samplers**

written by the user from the building blocks (`run_sampler()`'s `step()`) are checked with `step`, `prior`, `simulate` and `monitor` (Geweke test): `prior()` returns a state drawn from the prior (all parameters and the person parameters), `simulate(s)` the data list given the state, and `step(s, data)` is the sampler. Keep the model small (10-20 persons, a few items) and the priors proper and not too wide, so that the chain mixes. The test finds wrong full conditionals and forgotten or wrong prior terms, which summaries of a fit cannot show; its power is that of a test on a small model (an error that moves the posterior by a fraction of a posterior SD can pass), so passing is evidence, not proof. This checks the code of the sampler, not the convergence of your own analysis: for that use `convergence()` and `plot(fit, "trace")`.

Value

A data frame of class `rtirt_check` with one row per comparison (Geweke: parameter, probability, share in the chain, effective sample size, z, p-value, Holm-adjusted p-value; SBC: parameter, chi-square statistic, p-value, Holm-adjusted p-value) and attributes `method`, `passed` (all Holm-adjusted p-values above 0.01), `draws` (Geweke chain) or `ranks` (SBC).

References

Cook, S. R., Gelman, A., & Rubin, D. B. (2006). Validation of software for Bayesian models using posterior quantiles. *Journal of Computational and Graphical Statistics, 15*, 675-692.

Geweke, J. (2004). Getting it right: Joint distribution tests of posterior simulators. *Journal of the American Statistical Association, 99*, 799-804.

Modrák, M., Moon, A. H., Kim, S., Bürkner, P., Huurre, N., Faltejsková, K., Gelman, A., & Vehtari, A. (2025). Simulation-based calibration checking for Bayesian computation: The choice of test quantities shapes sensitivity. *Bayesian Analysis, 20*(2).

Talts, S., Betancourt, M., Simpson, D., Vehtari, A., & Gelman, A. (2018). Validating Bayesian inference algorithms with simulation-based calibration. arXiv:1804.06788.

Examples

```
cond <- set_cond(n_subj = 100, n_item = 3)
dat <- sim_data(cond, sim_para(cond, "cross"), "cross")
fit <- ecm(dat, model = "cross", se = FALSE) # the model to check (its fit is not used)
narrow <- rtirt_priors(lambda = c(mean = 0, sd = 1)) # any proper prior; narrow mixes fast
check_sampler(fit, n_iter = 1e5, priors = narrow, seed = 1) # Geweke
check_sampler(fit, method = "sbc", n_rep = 100, priors = narrow, seed = 1)
# a wrong prior in the sampler is detected
wrong <- rtirt_priors(lambda = c(mean = 0, sd = 1), a = c(mean = 1, sd = 0.3))
check_sampler(fit, n_iter = 2e4, priors = narrow, sampler_priors = wrong, seed = 1)
```

---

check_vi *Check a CAVI fit (deprecated)*

Description

Deprecated with `vi()`.

Usage

```
check_vi(
  model,
  steps = NULL,
  latent = NULL,
  nodes = NULL,
  n_start = 5,
  sweeps = 20,
  seed = NULL,
  quadrature = c("adaptive", "ba81")
)
```

Arguments

model	A block_model().
steps	Steps, as for gibbs() (the same list can be used for both); by default Gibbs (conditional maximization) where possible and MALA (gradient) otherwise.
latent	Parameters integrated out in the E-step (one to three [person] parameters with a normal prior).
nodes	Quadrature points per latent variable (default 21, 11 or 7 for one, two or three); one number, or one per latent variable (named, e.g. c(theta = 11, zeta = 5)).
n_start	Number of starting values.
sweeps	Sweeps per start in check 1.
seed	Random seed.
quadrature	"adaptive" (default): q(person) on nodes centred and scaled per person. "ba81": the same nodes for every person (Bock & Aitkin, 1981; default 41, 21 or 11 per latent variable), with the logistic parts summed over persons at each node, as in ecm(); faster for many persons (the ELBO is that of the common nodes; use the adaptive nodes when a latent variable has a narrow posterior).

Details

The CAVI counterpart of check_ecm() and the Geweke check: coordinate ascent is right when

1. **the ELBO never decreases** from one sweep to the next (from n_start starting values drawn from the declared priors, on a fixed grid); and
2. **the fit is a maximum of the ELBO**: moving the mean of any q (normal factors) a little up or down lowers it.

Value

A data frame with one row per check (class em_check, printed like check_ecm()).

`ci_test` *Test of conditional independence between responses and response times*

Description

The Lagrange multiplier (score) test of van der Linden & Glas (2010): under conditional independence the distribution of the log time on item i does not depend on whether the response is correct, given ability and speed. The alternative shifts the log times of correct responses, $\log t_{ij} = \lambda_i + \delta_i y_{ij} - \zeta_j[-\rho_i\theta_j] + e_{ij}$, and the test is of $\delta_i = 0$ from the maximum likelihood fit of the model without the shift (`ecm()`). The score of δ_i for person j is $y_{ij}E[\log t_{ij} - \lambda_i + \zeta_j(+\rho_i\theta_j)|\text{data}_j]/\sigma_i^2$ (Fisher identity). The information is the observed information of the model extended by the shifts (central differences of the analytic score, as for the standard errors of `ecm()`) with the parameters of the model projected out, so the statistic is chi-square(1) per item, and chi-square(K) for all items together. (The outer product of the casewise scores is simpler but rejects too often: about .065 per item and .10 overall at the 5% level with 500 persons and 10 items.) A positive `delta` means that correct responses took longer than incorrect ones.

Usage

```
ci_test(object, ...)
```

Arguments

`object`	A model fitted by `ecm()` (normal models with response times).
`...`	Unused.

Value

A data frame of class `rtirt_ci_test` per item: `item`, `delta` (one-step estimate of the shift), `statistic`, `p.value`, `p.holm`; attribute `overall` (joint test of all items).

References

van der Linden, W. J., & Glas, C. A. W. (2010). Statistical tests of conditional independence between responses and/or response times on test items. *Psychometrika, 75*, 120-139.

Examples

```
cond <- set_cond(n_subj = 500, n_item = 8)
d <- sim_data(cond, sim_para(cond, "cross"), "cross")
ci_test(ecm(d, model = "null", se = FALSE))
```

compare_para *Compare estimates with true values*

Description

As `comparePara()` in Julia (for simulations with `true_para`): bias, RMSE and correlation of the posterior means (or of the ECM estimates) for one parameter block.

Usage

```
compare_para(object, par = "a")
```

Arguments

`object`	A sampled model with `true_para`.
`par`	Parameter name, e.g. `"a"`, `"b"`, `"ability"`, `"lambda"`, `"rho"`, `"b_ability"`, `"beta"` (the Julia names `"theta"`, `"b_theta"` and, for `rtirt_latent`, `"rho"` are accepted).

compose *Helpers for composing samplers*

Description

General steps that come back in many samplers written from the blocks (blocks); with them the samplers of the examples `ex08`-`ex12` (testlet, multidimensional, conditional dependence, LNIRT, variable speed) are a few lines per step. Persons j are the rows and items i the columns of the matrices.

Usage

```
calc_normal_part(x, z, w = 1, rest = 0, by = c("item", "person"))

draw_mvn(prec, num, lower = -Inf, upper = Inf, max_tries = 100)

draw_items_mh(
  Y,
  theta,
  a,
  b,
  prior_mean,
  prior_prec,
  link = c("logit", "probit"),
  form = c("difficulty", "intercept")
)
```

```
draw_invwishart(df, scale)

draw_mean_mvn(X, Sigma, mean0, cov0)

draw_cor(x1, x2, cor, steps = 5, step = 0.05, method = c("rw", "mala"))

calc_person_fit(L, fitted, sigma2)
```

Arguments

x, z, w, rest	Numbers or N x K matrices (persons x items).
by	"item" or "person".
prec, num	Precision matrix and linear term (or an array of precisions and a matrix of linear terms, one slice / column per draw).
lower, upper	Bounds of the truncation box (recycled to the dimension).
max_tries	Rejection attempts per draw.
Y	Responses (0/1, NA for missing), N x K.
theta	Abilities.
a, b	Current item parameters.
prior_mean, prior_prec	Prior mean (K x 2 or length 2) and prior precision (2 x 2) of (a, b).
link	"logit" or "probit".
form	"difficulty" or "intercept".
df, scale	Degrees of freedom and scale matrix.
X	Matrix whose rows share the mean.
Sigma	Covariance of the rows.
mean0, cov0	Prior mean and covariance of the common mean.
x1, x2	The two variables (standardized by the model).
cor	Current correlation.
steps, step	Number and size of the Metropolis steps.
method	"rw" (random walk) or "mala".
L, fitted, sigma2	Log times, their fitted means (N x K) and the residual variances (per item).

Details

- `calc_normal_part(x, z, w, rest, by)`: the part of a full conditional that comes from normal (pseudo-)observations $z_{ij} = x_{ij}\beta + \text{rest}_{ij} + e_{ij}$, $e_{ij} \sim N(0, 1/w_{ij})$, for a parameter β of each item (`by = "item"`, sums over persons) or of each person (`by = "person"`, sums over items): precision $\sum wx^2$ and linear term $\sum wx(z - \text{rest})$. Add the prior and draw with `draw_gauss()` or `draw_tnorm()`. Pólya–Gamma: `z = (Y - 1/2) / omega`, `w = omega`; probit: `z` from `draw_probit_z()`, `w = 1`; log times: `z = log t`, `w = 1 / sigma2`. Missing `z` (NA) are skipped.

- `draw_mvn(prec, num, lower, upper)`: multivariate normal $N(Q^{-1}h, Q^{-1})$ from the precision `prec` (Q) and linear term `num` (h); `prec` can be an array p x p x n and `num` a p x n matrix for n independent draws (one per item or person), optionally truncated to the box `[lower, upper]` (rejection; a slice that fails is NA).
- `draw_items_mh(Y, theta, a, b, prior_mean, prior_prec, link, form)`: item parameters (a, b) with the latent responses integrated out, Metropolis–Hastings with a Fisher-scoring proposal (the cure for the slow mixing of easy and difficult items under data augmentation); logit or probit, `form` = `"difficulty"` for a (theta - b) or `"intercept"` for a theta - b; prior N((a, b); `prior_mean` (a row per item or one row), `prior_prec`^-1). Returns `list(a, b)` with the attribute `accepted` (as `draw_items_joint()`); draw the latent responses anew afterwards.
- `draw_invwishart(df, scale)`: inverse Wishart.
- `draw_mean_mvn(X, Sigma, mean0, cov0)`: the common mean of the rows of `X` ~ N(mu, Sigma) with prior mu ~ N(mean0, cov0) (hyperparameters of item- or person-level distributions).
- `draw_cor(x1, x2, cor, steps, step, method)`: the correlation of two standard-normal variables (unit variances, the identification of several abilities), `steps` Metropolis steps on atanh(cor) with a uniform prior: random walk (`"rw"`) or MALA (`"mala"`, see `draw_mala()`).
- `calc_person_fit(L, fitted, sigma2)`: the RT person-fit statistic of Marianti et al. (2014) $l_j^t = \sum_i (\log t_{ij} - \text{fitted}_{ij})^2/\sigma_i^2$ per person, with the number of observed times `df` (missing times are skipped); chi-square with `df` degrees of freedom.

Value

`calc_normal_part()`: a normal part (`prec`, `num`; see normal_parts); `draw_mvn()`: a vector or a p x n matrix; `draw_items_mh()`: a list; `draw_invwishart()`: a matrix; `draw_mean_mvn()`: a vector; `draw_cor()`: a number with attribute `accepted`; `calc_person_fit()`: a data frame.

References

Marianti, S., Fox, J.-P., Avetisyan, M., Veldkamp, B. P., & Tijmstra, J. (2014). Testing for aberrant behavior in response time modeling. *Journal of Educational and Behavioral Statistics, 39*, 426-451.

Examples

```
set.seed(1)
N <- 300; K <- 5; theta <- rnorm(N); a <- rep(1.2, K); b <- seq(-1, 1, length.out = K)
Y <- matrix(rbinom(N * K, 1, plogis(sweep(outer(theta, a), 2, a * b))), N)
om <- draw_omega(a, b, theta)
# b_i | omega: eta = a (theta - b) enters the PG pseudo-observations with x = -a
g <- calc_normal_part(matrix(-a, N, K, byrow = TRUE), (Y - 0.5) / om, om, rest = outer(theta, a))
draw_gauss(1 / 9 + g$prec, g$num)                       # prior N(0, 3^2)
# (a, b) jointly with the latent responses integrated out
draw_items_mh(Y, theta, a, b, prior_mean = c(1, 0), prior_prec = diag(c(1, 1 / 9)))
draw_mvn(diag(2) * 4, c(1, 2), lower = 0)
```

convergence *Convergence check*

Description

The share of item and structural parameters with effective sample size above `ess` and R-hat below `rhat` (over independent chains), and the worst parameters. With `diagnostics = "rank"` (see `estimates()`) both the bulk and the tail ESS must exceed `ess`, and the default R-hat threshold is 1.01 (Vehtari et al., 2021) instead of 1.05. Parameters that do not vary (fixed ones) are left out. With one chain R-hat cannot be computed and only the ESS is checked. `print()` of a `gibbs()` fit warns with the same criteria (default thresholds) when a parameter fails them. Julia: `checkConvergence()`.

Usage

```
convergence(object, ...)

## S3 method for class 'rtirt'
convergence(
  object,
  ess = 400,
  rhat = NULL,
  diagnostics = c("classic", "rank"),
  ...
)

## S3 method for class 'rtirt_qset'
convergence(object, ...)

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

Arguments

`object`	A sampled model, an `rtirt_qset`, or an `ecm()` fit (whether ECM converged).
`...`	Unused.
`ess`, `rhat`	Thresholds (`rhat` defaults to 1.05, or 1.01 with `diagnostics = "rank"`).
`diagnostics`	`"classic"` or `"rank"`, as in `estimates()`.
`x`	An `rtirt_convergence` object.

Details

For several quantile levels (an `rtirt_qset` from `gibbs(..., quantile = c(.25, .75))`) each level is checked; the model of one level is `fits$models[[k]]`.

Value

An object of class `rtirt_convergence`, printed by its `print()` method: `share` (share of parameters that pass), `worst` (the five parameters with the largest R-hat, or the smallest ESS with one chain), `n` (parameters checked), `n_chain` and the thresholds; for an ECM fit `converged` and `iterations`; for an `rtirt_qset` one such object per level (`levels`) with `quantile`.

Examples

```
cond <- set_cond(n_subj = 200, n_item = 6)
dat <- sim_data(cond, sim_para(cond, "null"), "null")
fit <- gibbs(dat, model = "null", n_iter = 1000, n_chain = 2, seed = 1)
check <- convergence(fit)   # printed only when not assigned
check
check$share
```

`dic` *Deviance information criterion*

Description

As `getDic(MCMC)` in Julia: $\bar{D}$ is the posterior mean of the complete-data deviance (after burn-in; the Julia version averaged over the burn-in as well), $\hat{D}$ the deviance at the posterior means, $p_D = \bar{D} - \hat{D}$. The deviance conditions on the person parameters (and, for the ALD models, on the mixing variables), so DIC values are not comparable between mean and quantile models or across quantile levels.

Usage

```
dic(object, ...)

## S3 method for class 'rtirt'
dic(object, ...)
```

Arguments

`object`	A sampled model.
`...`	Unused.

`draw_shift` *Exact shift of parameters that move together*

Description

Some directions of a model are identified only by the priors: adding the same constant to the abilities, the difficulties and the intercept of a latent regression leaves the likelihood of a 2PL unchanged. A Gibbs sampler moves along such a direction very slowly. `draw_shift()` draws the shift c of the move $x_k \leftarrow x_k + c\,d_k$ exactly from its conditional (a generalized Gibbs step; Liu & Sabatti, 2000): only the normal priors of the shifted parameters depend on c. Parameters listed in `values` without a prior are shifted with a prior that moves along (e.g. abilities whose prior mean contains the shifted intercept).

Usage

```
draw_shift(values, prior, along = list(), lower = list())
```

Arguments

`values`	Named list of the current values to shift.
`prior`	Named list of normal priors `c(mean, sd)` (or `list(mean = , sd = )` with one value per element) for the shifted parameters whose prior does not move along.
`along`	Named list of direction multipliers (default 1).
`lower`	Named list of lower bounds of truncated priors (e.g. positive loadings): the shift is drawn from its conditional truncated so that every bound holds.

Details

The caller states that the likelihood is unchanged by the shift. Common cases:

model	shift	`values`, `prior`
2PL, theta ~ N(g0 + x'g, 1)	theta + c, b + c, g0 + c	`list(theta, b, g0)`, priors of `b` and `g0`
log T = lambda - zeta	lambda + c, zeta + c	`list(lambda, zeta)`, priors of both (speed: N(m_j, 1 / w_j)), `lower =`
intercept form a theta - d	theta + c, d + a c	`along = list(d = a)`

Value

`values` shifted, with attribute `shift`.

References

Liu, J. S., & Sabatti, C. (2000). Generalised Gibbs sampler and multigrid Monte Carlo for Bayesian computation. *Biometrika, 87*, 353-369.

Examples

```
s <- draw_shift(list(theta = rnorm(5), b = c(-1, 0, 1), g0 = 0.2),
                prior = list(b = c(0, 3), g0 = c(0, 10)))
attr(s, "shift")
```

`ecm` *Marginal maximum likelihood (ECM)*

Description

Fits a built-in normal model by maximum marginal likelihood: the EM algorithm of Bock and Aitkin (1981) with ability integrated by Gauss-Hermite quadrature and speed in closed form. The item step uses the Polya-Gamma bound of the logistic likelihood (Polson, Scott & Windle, 2013), so every step is closed form and the log-likelihood never decreases; SQUAREM speeds up convergence. Standard errors come from the observed information (the numerical derivative of the analytic score).

Usage

```
ecm(data, ...)

## S3 method for class 'input_data'
ecm(
  data,
  model = NULL,
  itemtype = "2pl",
  speed_var = c("free", "fixed"),
  quantile = NULL,
  cov = NULL,
  prior = c("ml", "map"),
  priors = NULL,
  se = TRUE,
  control = ecm_control(),
  ...
)
```

Arguments

`data`	Data from `input_data()`.
`...`	Not used: any other argument is an error before the fit starts. A setting of `ecm_control()` (`nodes`, `max_iter`, `tol`, `accelerate`, `verbose`) gets the `control = ecm_control(...)` form in the message, and a misspelled name the closest argument.
`model`	The model by name (case does not matter): • `"irt"`: 2PL, no response times (with covariates: ability regressed on them).

- "null": responses and times; ability and speed correlated (van der Linden, 2007).
- "latreg": as "null", with ability and speed regressed on the covariates.
- "latent": speed regressed on ability and the covariates.
- "cross": item-specific effects of ability on the log times (cross-relations).

A misspelled name gets the closest one in the error message. "mixture" (Wang & Xu, 2015) is not available yet.

itemtype	"2pl" or "1pl" ("grm" is not available yet); one value, or one per item (all equal for now).
speed_var	"free" (default) or "fixed" (variance of speed 1; "null", "latreg", "cross").
quantile	NULL (the normal model), or one or more quantile levels in (0, 1) for "latent" and "cross" (see Quantile models).
cov	Names of the covariates of data to use (default all); models without covariates ignore them.
prior	"ml" (maximum likelihood) or "map": "map" adds the item priors of priors (a, b, lambda, sigma2t and rho; by default those of gibbs(), see rtirt_priors()): the posterior mode in the item parameters, for small samples. The structural parameters stay maximum likelihood.
priors	Prior distributions for prior = "map", from rtirt_priors().
se	Compute standard errors.
control	Further settings, from ecm_control(), e.g. ecm(dat, model = "cross", control = ecm_control(nodes = 61)).

Details

Quantile models. With quantile = (models "latent" and "cross", as in gibbs()) ecm() maximizes the marginal ALD working likelihood (Yu & Moyeed, 2001): the q-quantile of the log times ("cross") or of speed given ability and the covariates ("latent") is linear in the parameters. Ability is integrated on the nodes and speed in closed form, so the working log-likelihood is smooth. The E-step and the item step are those of the normal models; the response-time and structural parameters are updated by nlminb() with the analytic gradient (two iterations per ECM step, which never lowers the working log-likelihood). The fit starts at the ECM fit of the mean model, moved to the q-quantile. The standard errors are sandwich (robust) standard errors, $H^{-1}JH^{-1}$ with J the sum of the outer products of the scores of the persons: the ALD is not the distribution of the data, so the observed information alone would be wrong. As in gibbs(), sigma2t of "cross" is the ALD scale of the log times and var_speed of "latent" the ALD scale of speed. The working logLik, AIC and BIC compare only fits at the same quantile; anova() refuses quantile fits. A vector of levels gives one fit per level (an rtirt_qset).

The parameterization and identification are those of gibbs(): ability has mean x'beta and variance 1, speed mean x'beta (no intercepts: the covariates are centred and the item parameters carry the location). The fit starts the Gibbs sampler:

```
fit_ecm <- ecm(dat, model = "cross")
fit_gibbs <- gibbs(dat, model = "cross", init = fit_ecm)
```

Small samples. `print()` and `summary()` warn when the estimates suggest that the data are too few for maximum likelihood: a difficulty with $|b| > 6$, a discrimination below 0.1 or above 10, or a standard error above 5 (or missing) for a, b or rho. These thresholds flag only clearly broken fits (with ability N(0, 1), $|b| > 6$ means almost everyone or no one solves the item, and an SE of 5 a 95% interval about 20 wide); `prior = "map"` or `gibbs()` regularize such fits.

Value

A fitted model with `ecm`: `estimates` (est, se, z, Wald 95% interval on the log scale for positive parameters and the Fisher-z scale for the correlation), `logLik`, `npar`, `AIC`, `BIC`, `iterations`, `converged`, `trace` (log-likelihood per iteration), `vcov`, `person` (EAP and posterior SD) and `secs`. Accessors: `print`, `summary`, `coef`, `estimates()`, `scores()`, `reliability()`, `fit_indices()`, `logLik`, `vcov`, `AIC`, `BIC`.

References

Bock, R. D., & Aitkin, M. (1981). Marginal maximum likelihood estimation of item parameters: Application of an EM algorithm. *Psychometrika*, 46, 443-459.

Polson, N. G., Scott, J. G., & Windle, J. (2013). Bayesian inference for logistic models using Polya-Gamma latent variables. *JASA*, 108, 1339-1349.

White, H. (1982). Maximum likelihood estimation of misspecified models. *Econometrica*, 50, 1-25.

Yu, K., & Moyeed, R. A. (2001). Bayesian quantile regression. *Statistics & Probability Letters*, 54, 437-447.

Varadhan, R., & Roland, C. (2008). Simple and globally convergent methods for accelerating the convergence of any EM algorithm. *Scandinavian Journal of Statistics*, 35, 335-353.

Examples

```
cond <- set_cond(n_subj = 300, n_item = 8)
dat <- sim_data(cond, sim_para(cond, "cross"), "cross")
fit <- ecm(dat, model = "cross")
fit
head(estimates(fit))

fit_q <- ecm(dat, model = "cross", quantile = 0.9) # the 0.9-quantile of the log times
estimates(fit_q, pars = "^rho")
```

`ecm.block_model` *ECM for a declared model (deprecated)*

Description

Deprecated with `block_model()`: use the built-in models, `ecm(data, model = ...)`.

Usage

```
## S3 method for class 'block_model'
ecm(
  data,
  steps = NULL,
  latent = NULL,
  nodes = NULL,
  prior = c("map", "ml"),
  init = NULL,
  cycles = 6,
  max_iter = 1000,
  tol = 1e-10,
  se = TRUE,
  analytic = TRUE,
  quadrature = c("adaptive", "ba81"),
  ...
)

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

## S3 method for class 'block_model_ecm'
algorithm(object, format = c("unicode", "latex"), ...)
```

Arguments

`data`	A `block_model()`.
`steps`	Steps, as for `gibbs()` (the same list can be used for both); by default Gibbs (conditional maximization) where possible and MALA (gradient) otherwise.
`latent`	Parameters integrated out in the E-step (one to three `[person]` parameters with a normal prior).
`nodes`	Quadrature points per latent variable (default 21, 11 or 7 for one, two or three); one number, or one per latent variable (named, e.g. `c(theta = 11, zeta = 5)`).
`prior`	`"map"` (posterior mode) or `"ml"` (maximum marginal likelihood).
`init`	Optional starting values (named list).
`cycles`	Maximum number of recentring rounds of the quadrature.
`max_iter, tol`	Passed to `run_em()`.
`se`	Standard errors from the Hessian of the objective.
`analytic`	Integrate latent variables that enter the model only through normal parts, linearly (e.g. speed in a log-normal RT model), exactly given the others instead of on the grid.
`quadrature`	`"adaptive"` (default): nodes centred and scaled per person. `"ba81"`: the same nodes for every person (Bock & Aitkin, 1981; default 41, 21 or 11 per latent variable), centred at the marginal posterior. A part of the data that depends on the persons only through the variables on the grid is then summed over persons at each node ("artificial data"), so the E-step is a matrix product and the

	M-step works on nodes x items: faster for many persons. A latent variable whose posterior is narrow compared with its spread over persons needs many common nodes; the fit compares its objective with adaptive nodes at the estimates (`fit$adaptive_gap`) and warns when they differ by more than 0.5 (with `analytic = FALSE` speed is then on the common grid too, and its posterior is narrow when there are many items: use `analytic = TRUE`). Still ECM: only the numerical integration of the E-step changes, the conditional maximizations are the same.
`...`	Unused.
`x`	A fit of `ecm()`.
`object`	A fit of `ecm()`.
`format`	`"unicode"` (default) or `"latex"` (an `aligned` block).

Details

Fits a `block_model()` by EM with the same steps as `gibbs()`. Gibbs and EM share the full conditionals of the model: a step that a sampler uses to **draw** a parameter is used by EM to move the parameter to the **mode** of its full conditional averaged over the latent variables.

- **E-step:** the person parameters (`latent`, by default every `[person]` parameter) are integrated out by adaptive Gauss–Hermite quadrature: a product grid of `nodes` points per latent variable, centred at each person's posterior mean and scaled by the posterior SD. The centres are fixed while EM runs (so the objective is one fixed function and EM never lowers it), then recentred, for up to `cycles` rounds. The Pólya–Gamma variables of a logistic part are replaced by their expectation $E[\omega] = \tanh(\eta/2)/(2\eta)$ at each node (a minorizer: the step still never lowers the objective).
- `step_gibbs()` becomes a conditional maximization: the normal part of the parameter is summed over the model and over the nodes (weighted by the posterior of each person), and the parameter moves to its mode m/p (clipped to the bounds); a variance moves to the mode of its inverse-gamma conditional (`prior = "ml"`: β/α).
- `step_mala()` becomes a few gradient-ascent steps with step halving on the same weighted objective (generalized EM), on the log scale for bounded parameters (no Jacobian: a mode does not need one).
- `step_shift()` is skipped: the latent variables are integrated out.

The objective is the log marginal likelihood (`prior = "ml"`; the priors of the latent variables stay, they define the marginal likelihood) or the log marginal posterior (`prior = "map"`, the default: also the priors of the other parameters). Standard errors come from its Hessian (Louis' formula: expected complete-data Hessian plus the covariance of the complete-data score, from the derivatives of the declared expressions); the posterior means and SDs of the latent variables are in `fit$latent`. EM runs through `run_em()` (SQUAREM, the checks of `run_em()`). Use `check_ecm()` to check the steps.

Value

A `block_em` (see `run_em()`) with `latent` (posterior means and SDs per person), `model`, `steps` and `cycles`; `algorithm(fit)` lists the E- and M-steps.

Theory of each step

- **ECM.** EM needs only the conditional expectations of the complete-data sufficient statistics (Dempster, Laird & Rubin, 1977, Sec. 2) and never lowers the objective when every M-step raises Q (their Theorem 1, generalized EM); the M-step may be a sequence of conditional maximizations (Meng & Rubin, 1993).
- **Adaptive quadrature.** Nodes centred and scaled at each person's posterior (Naylor & Smith, 1982; Liu & Pierce, 1994; Bock & Gibbons, 2021, Sec. 6.4). Schilling & Bock (2005) recentre at every EM iteration; here the centres are kept fixed within a round and recentred between rounds, as Bock & Gibbons (2021, pp. 210–211) suggest to save time and as in the pseudo-adaptive rule of Rizopoulos (2012). With fixed centres the approximated objective is one fixed function, so the monotonicity of Dempster et al. (1977, Theorem 1) applies to it (our argument; no source states it).
- `quadrature = "ba81"`**.** Common nodes and expected counts per node ("artificial data"): Bock & Aitkin (1981); Bock & Gibbons (2021, Sec. 6.4, Eqs. 6.17–6.22).
- **Logistic parts.** $E[\omega] = \tanh(\eta/2)/(2\eta)$ (Polson, Scott & Windle, 2013, Sec. 2.2) makes the M-step an exact EM step for the Pólya–Gamma augmented model (Scott & Sun, 2013); it equals the Jaakkola–Jordan bound, a minorize–maximize step that never lowers the likelihood (Durante & Rigon, 2019, Appendix).
- **Exact integration (`analytic`).** A latent variable u that enters only normal parts linearly, with variances free of u, has a normal conditional $u \mid \cdot \sim N(h/p, 1/p)$ (completing the square; Bishop, 2006, Sec. 2.3; for speed in the hierarchical RT model, van der Linden, 2007, Eq. 42), and the integral over u is the integrand at the mode times $\sqrt{2\pi/p}$, exact for a quadratic log integrand (Bishop, 2006, Eq. 4.135). The RT-IRT marginal likelihood papers we found integrate speed numerically together with ability (Glas & van der Linden, 2010; Molenaar & Bolsinova, 2017); integrating it exactly is our reduction, valid only under the condition above (checked from the declaration; not for heteroscedastic or skewed speed).
- **Exact variables in the M-step.** When every part of an exact variable is linear in all latent variables jointly, with variances free of them, these parts are quadratic in the latent variables, and the expectation of a quadratic depends only on the mean and covariance ($E[x'Ax] = \mathrm{tr}(A\Sigma) + \mu' A\mu$, for any distribution). The M-step therefore uses each person's posterior mean and covariance of the latent variables through the 2L symmetric points $m \pm \sqrt{L}$ x columns of the Cholesky factor, weight 1/(2L) each, which have exactly that mean and covariance (Julier & Uhlmann, 2004, Eq. 11). Logistic parts stay on the nodes.
- **Standard errors.** Louis (1982): observed information = expected complete-data information minus the covariance of the complete-data score. Under `prior = "map"` the log prior of the fixed parameters does not involve the latent variables, so its Hessian is added to that of the log marginal likelihood (Chalmers, 2018, notes that how priors enter Louis' form needs saying). The squared scores are quartic in an exact variable, so three Gauss–Hermite points per exact variable are used (n points are exact to degree 2n - 1; DLMF Sec. 3.5(v)).

References

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Bock, R. D., & Aitkin, M. (1981). Marginal maximum likelihood estimation of item parameters: Application of an EM algorithm. *Psychometrika, 46*(4), 443–459. doi:10.1007/BF02293801

Bock, R. D., & Gibbons, R. D. (2021). *Item response theory*. Wiley. doi:10.1002/9781119716723

Chalmers, R. P. (2018). Numerical approximation of the observed information matrix with Oakes' identity. *British Journal of Mathematical and Statistical Psychology, 71*(3), 415–436. doi:10.1111/bmsp.12127

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B, 39*(1), 1–22. doi:10.1111/j.2517-6161.1977.tb01600.x

DLMF. *NIST Digital Library of Mathematical Functions*, Sec. 3.5(v), Gauss quadrature. https://dlmf.nist.gov/3.5

Durante, D., & Rigon, T. (2019). Conditionally conjugate mean-field variational Bayes for logistic models. *Statistical Science, 34*(3), 472–485. doi:10.1214/19STS712

Glas, C. A. W., & van der Linden, W. J. (2010). Marginal likelihood inference for a model for item responses and response times. *British Journal of Mathematical and Statistical Psychology, 63*(3), 603–626. doi:10.1348/000711009X481360

Julier, S. J., & Uhlmann, J. K. (2004). Unscented filtering and nonlinear estimation. *Proceedings of the IEEE, 92*(3), 401–422. doi:10.1109/JPROC.2003.823141

Liu, Q., & Pierce, D. A. (1994). A note on Gauss–Hermite quadrature. *Biometrika, 81*(3), 624–629. doi:10.1093/biomet/81.3.624

Louis, T. A. (1982). Finding the observed information matrix when using the EM algorithm. *Journal of the Royal Statistical Society B, 44*(2), 226–233. doi:10.1111/j.25176161.1982.tb01203.x

Meng, X.-L., & Rubin, D. B. (1993). Maximum likelihood estimation via the ECM algorithm: A general framework. *Biometrika, 80*(2), 267–278. doi:10.1093/biomet/80.2.267

Molenaar, D., & Bolsinova, M. (2017). A heteroscedastic generalized linear model with a non-normal speed factor for responses and response times. *British Journal of Mathematical and Statistical Psychology, 70*(2), 297–316. doi:10.1111/bmsp.12087

Naylor, J. C., & Smith, A. F. M. (1982). Applications of a method for the efficient computation of posterior distributions. *Applied Statistics, 31*(3), 214–225. doi:10.2307/2347995

Polson, N. G., Scott, J. G., & Windle, J. (2013). Bayesian inference for logistic models using Pólya–Gamma latent variables. *Journal of the American Statistical Association, 108*(504), 1339–1349. doi:10.1080/01621459.2013.829001

Rizopoulos, D. (2012). Fast fitting of joint models for longitudinal and event time data using a pseudo-adaptive Gaussian quadrature rule. *Computational Statistics & Data Analysis, 56*(3), 491–501. doi:10.1016/j.csda.2011.09.007

Schilling, S., & Bock, R. D. (2005). High-dimensional maximum marginal likelihood item factor analysis by adaptive quadrature. *Psychometrika, 70*(3), 533–555. doi:10.1007/s113360031141x

Scott, J. G., & Sun, L. (2013). Expectation-maximization for logistic regression. arXiv:1306.0040.

van der Linden, W. J. (2007). A hierarchical framework for modeling speed and accuracy on test items. *Psychometrika, 72*(3), 287–308. doi:10.1007/s113360061478z

Examples

```
set.seed(1); N <- 300; K <- 6
th <- rnorm(N); a0 <- runif(K, 0.8, 2); b0 <- seq(-1.5, 1.5, length.out = K)
Y <- matrix(rbinom(N * K, 1, plogis(sweep(outer(th, a0), 2, a0 * b0))), N)
m <- block_model(Y = Y,
  theta[person] ~ normal(0, 1),
```

```
  a[item] ~ normal(1, 1, lower = 0),
  b[item] ~ normal(0, 3),
  eta <- a * (theta - b),
  Y ~ bernoulli_logit(eta))
fit <- ecm(m, list(step_gibbs(a, b)), prior = "ml")
fit
algorithm(fit)
```

`ecm_control` *Settings of the engines*

Description

The settings of `ecm()` and `gibbs()` that are rarely changed.

Usage

```
ecm_control(
  max_iter = 1000,
  tol = 1e-10,
  nodes = 41,
  accelerate = TRUE,
  verbose = FALSE,
  ...
)

gibbs_control(
  n_burnin = NULL,
  n_thin = 1,
  collapse = FALSE,
  fit_indices = TRUE,
  verbose = TRUE,
  ...
)

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

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

Arguments

`max_iter`	Maximum number of ECM (SQUAREM) iterations.
`tol`	Convergence: relative change of the log-likelihood below `tol` and largest parameter change (unconstrained scale) below `sqrt(tol)`.
`nodes`	Gauss-Hermite nodes for ability.

accelerate	Use SQUAREM.
verbose	ecm_control(): print the log-likelihood every 10 iterations; gibbs_control(): print the progress: a line when sampling starts, one per chain with its time and the time left (the chains run one after the other), and one before the marginal log-likelihood of fit_indices is computed.
...	Not used: a name that is not a setting is an error that names the closest setting, or the function it belongs to (gibbs_control(n_iter = 200): an argument of gibbs()).
n_burnin	Burn-in per chain (default n_iter / 2).
n_thin	Thinning of the stored draws.
collapse	FALSE (default): the Gibbs sampler of ExtendedRtIrtModeling.jl, plus the two group moves. TRUE: a partially collapsed Gibbs sampler (van Dyk & Park, 2008). Given the Polya-Gamma (and ALD) variables the model is linear-Gaussian, so speed is integrated out in closed form when ability, the RT item parameters (lambda, and rho of "cross") and, in "latent", the structural regression are drawn, and speed is drawn after them; in "irt" the regression coefficients are drawn with ability integrated out. The 2PL item parameters (a, b) are drawn jointly with the Polya-Gamma variables integrated out (a Metropolis-Hastings step with a Fisher-scoring proposal; the acceptance rate is in post$ab_accept). Same posterior, fewer slow directions.
fit_indices	Compute the marginal log-likelihood after sampling (for DIC, WAIC and PSIS-LOO in fit_indices(); see marginal_loglik()). FALSE skips it (it can take seconds to a minute for large "cross" quantile models); fit_indices() then computes it on request.
x	Settings from ecm_control() or gibbs_control().

Value

A list of settings of class ecm_control or gibbs_control; ecm() refuses a gibbs_control() and gibbs() an ecm_control(). A plain list with the names of the settings is also accepted.

Examples

```
fit_settings <- gibbs_control(n_burnin = 500, collapse = TRUE)
fit_settings
```

---

em_components *Components of the EM algorithm*

Description

The steps of ecm() as functions that can be combined into an EM for another model (the counterpart of the sampling blocks, blocks and compose: draw_*() draws a block from its full conditional, update_*() maximizes, or takes a monotone MM step, in the same block).

Usage

```
calc_nodes(Q = 41)

calc_loglik_nodes(Y, a, b, theta)

calc_normal_given_nodes(o, w, m, v)

calc_node_weights(logL, logw = 0)

calc_artificial_data(W, Y)

calc_item_sums(Y, a, b, theta, W, score = FALSE)

update_items_logistic(S, T0, T1, a, b, prior_a = NULL, prior_b = NULL)

em_squarem(
  x,
  Fmap,
  max_iter = 1000,
  tol = 1e-10,
  accelerate = TRUE,
  progress = NULL
)

calc_hessian(score, x, h = 1e-04)
```

Arguments

`Q`	Number of nodes.
`Y`	Responses (N x K, 0/1).
`a`, `b`	Item parameters.
`theta`	Common nodes (a vector) or person-specific nodes (N x Q).
`o`, `w`, `m`, `v`	Pseudo-observations (N x K), their precisions (per item), prior means (N x Q or a vector of Q) and prior variance of the normal latent variable.
`logL`, `logw`	Log-likelihood at the nodes (N x Q) and log weights of the nodes.
`W`	Posterior weights of the nodes (N x Q).
`score`	Score sums instead of minorizer sums.
`S`, `T0`, `T1`	Item sums (a list with `S0`, `S1`, `S2`) and the totals of `y - 1/2`.
`prior_a`, `prior_b`	Normal priors of a (on (0, Inf)) and b (MAP), or `NULL` (maximum likelihood).
`x`	Starting value / point.
`Fmap`	Fixed-point map.
`max_iter`, `tol`, `accelerate`	Iteration limit, convergence tolerance (relative change of the objective and `sqrt(tol)` for the largest change of x), use SQUAREM.

progress	Optional function called as progress(iteration, objective).
h	Relative step of the central differences.

Details

E-step

- calc_nodes(Q): Gauss–Hermite nodes z and log weights logw of N(0, 1).
- calc_loglik_nodes(Y, a, b, theta): the 2PL log-likelihood of every person at every node, $\sum_i y_{ij}\eta - \log(1 + e^{\eta})$, $\eta = a_i(\theta - b_i)$, an N x Q matrix. theta is the vector of nodes when they are common to all persons (two matrix products, Bock–Aitkin), or an N x Q matrix of person-specific nodes (latent regressions; C++).
- calc_normal_given_nodes(o, w, m, v): a normal latent variable u (e.g. speed) with prior N(m, v) at each node (m: N x Q, or a vector of Q), observed through normal pseudo-observations $o_{ij} = u_j + e_{ij}$, $e_{ij} \sim N(0, 1/w_i)$: integrated in closed form. Returns logI (N x Q, add to the log-likelihood), the conditional mean (N x Q) and var of u.
- calc_node_weights(logL, logw): posterior weights W of the nodes (N x Q) and the log-likelihood ll of every person.
- calc_artificial_data(W, Y): Bock–Aitkin's expected counts: n (Q x K, persons at each node answering each item) and r (Q x K, correct answers).
- calc_item_sums(Y, a, b, theta, W, score): weighted item sums over persons and nodes, either the Pólya–Gamma minorizer sums $S_r = \sum W E[\omega|\eta]\theta^r$ (score = FALSE) or the score sums $\sum W(y - P)\theta^r$ (score = TRUE), r = 0, 1, 2; common nodes go through the artificial data.

M-step

- update_items_logistic(S, T0, T1, a, b, prior_a, prior_b): the Pólya–Gamma (Jaakkola–Jordan) minorizer step for 2PL items: a 2 x 2 weighted least-squares problem per item from the sums S of calc_item_sums() and $T_0 = \sum_j (y_{ij} - 1/2)$, $T_1 = \sum_j (y_{ij} - 1/2)E[\theta_j]$; the marginal log-likelihood never decreases. With prior_a = c(mean, sd) and prior_b = c(mean, sd, lower, upper) (or a list whose mean has one value per item) the conditional modes (MAP).

Driver and information

- em_squarem(x, Fmap, ...): SQUAREM (Varadhan & Roland, 2008; SqS3) around a fixed-point map Fmap(x) returning list(x = , obj =) (one EM step from x and the log-likelihood at x), with a fall-back to plain EM steps when the extrapolation lowers the objective.
- calc_hessian(score, x): the symmetrized numerical Jacobian of an analytic score function (central differences); $(-H)^{-1}$ is the covariance of the estimates.

Value

See Details.

References

Bock, R. D., & Aitkin, M. (1981). Marginal maximum likelihood estimation of item parameters: Application of an EM algorithm. *Psychometrika, 46*, 443-459.

Varadhan, R., & Roland, C. (2008). Simple and globally convergent methods for accelerating the convergence of any EM algorithm. *Scandinavian Journal of Statistics, 35*, 335-353.

Examples

```
set.seed(1)
N <- 500; K <- 10; a <- runif(K, 0.8, 2); b <- rnorm(K)
Y <- matrix(rbinom(N * K, 1, plogis(sweep(outer(rnorm(N), a), 2, a * b))), N)
# an EM for the 2PL from the components
nd <- calc_nodes(41); kap <- Y - 0.5
Fmap <- function(x) {
  a <- exp(x[1:K]); b <- x[K + 1:K]
  nw <- calc_node_weights(calc_loglik_nodes(Y, a, b, nd$z), nd$logw)
  S <- calc_item_sums(Y, a, b, nd$z, nw$W)
  new <- update_items_logistic(S, colSums(kap), colSums(kap * drop(nw$W %*% nd$z)), a, b)
  list(x = c(log(new$a), new$b), obj = sum(nw$ll))
}
fit <- em_squarem(c(rep(0, K), rep(0, K)), Fmap)
round(cbind(a, a_hat = exp(fit$x[1:K]), b, b_hat = fit$x[K + 1:K]), 2)
```

`equations` *Model equations*

Description

Writes the model as equations, with the parameter names of `summary()` and the priors of `gibbs()`. The default is Unicode text for the console; `format = "latex"` gives an `aligned` block for a paper or an appendix (e.g. `writeLines(as.character(equations(fit, "latex")), "model.tex")`). Works before estimation too: give the data and the model as in `ecm()` and `gibbs()`.

Usage

```
equations(object, ...)

## S3 method for class 'rtirt_equations'
as.character(x, ...)

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

## S3 method for class 'rtirt'
equations(object, format = c("unicode", "latex"), priors = TRUE, ...)

## S3 method for class 'rtirt_qset'
```

```
equations(object, format = c("unicode", "latex"), priors = TRUE, ...)

## S3 method for class 'input_data'
equations(
  object,
  model = NULL,
  itemtype = "2pl",
  speed_var = c("free", "fixed"),
  quantile = NULL,
  cov = NULL,
  intercept = FALSE,
  format = c("unicode", "latex"),
  priors = TRUE,
  ...
)
```

Arguments

`object`	A fit of `ecm()` or `gibbs()`, data from `input_data()` (with `model`), or a model object of the deprecated constructors (models) or an `rtirt_qset`.
`...`	Unused.
`x`	An `rtirt_equations` object.
`format`	`"unicode"` or `"latex"`.
`priors`	Include the priors (omitted for an `ecm()` fit, unless `prior = "map"`).
`model`, `itemtype`, `speed_var`, `quantile`, `cov`, `intercept`	For `input_data`: the model as in `ecm()` and `gibbs()`, so its equations can be read before estimation, e.g. `equations(dat, model = "cross", quantile = 0.25)`.

Value

An `rtirt_equations` object (printed); `as.character()` gives the lines.

Examples

```
cond <- set_cond(n_subj = 100, n_item = 5)
dat <- sim_data(cond, sim_para(cond, "cross"), "cross")
equations(dat, model = "cross")
equations(dat, model = "cross", quantile = 0.25, format = "latex")
```

`estfun.rtirt` *Casewise scores of an ecm() fit*

Description

`estfun()` returns, for a model fitted by `ecm()`, the matrix of casewise scores: one row per person, one column per parameter, the derivative of that person's marginal log-likelihood (ability and speed integrated out). The columns sum to zero at the estimates. The parameters are on the unconstrained scale used by `ecm()`: log a, b, lambda, log sigma2t, then the structural parameters (for `model = "latreg"` / `"null"` with free speed variance: the regression coefficients, the covariance `c` and the log residual variance `log_v` of speed given ability). These are the estimating functions used by `score_test()` and by `strucchange::gefp()`.

Usage

```
## S3 method for class 'rtirt'
estfun(x, ...)

## S3 method for class 'rtirt'
bread(x, ...)
```

Arguments

`x`	A model fitted by `ecm()` (with `prior = "ml"`).
`...`	Unused.

Value

A persons x parameters matrix.

`bread()`: the inverse of the average observed information (unconstrained scale), as in the 'sandwich' package.

`estimates` *Parameter estimates*

Description

`estimates()` returns the posterior summary of the item and structural parameters as a data frame: posterior mean (`est`), `sd`, 2.5% and 97.5% quantiles (`q025`, `q975`), R-hat and effective sample size (after burn-in, over the independent chains); `sig = "*"` when the 95% interval excludes 0. `sig` is given only for parameters where 0 is a meaningful value: difficulties `b`, regression coefficients `beta`, cross-relations `rho`, `cor_ability_speed` and `b_ability`. It is `NA` for discriminations, variances (`sigma2t`, `var_speed`), which are positive and whose intervals never contain 0, and time intensities `lambda`. `precis()` (the Julia name) returns the same table, printed in a plain layout (assigning it prints nothing); `as.data.frame()` gives it as a plain data frame, e.g. for ggplot2. `coef()` returns the posterior means as a named vector. For a model fitted by `ecm()` the table has the maximum likelihood estimates with standard errors (`est`, `se`, `z`, Wald 95% interval `q025`, `q975`); `z`, like `sig`, is `NA` where 0 is not a meaningful value, and `summary()` then shows no z or p.

Usage

```
estimates(object, ...)

## S3 method for class 'rtirt'
estimates(
  object,
  pars = NULL,
  method = c("auto", "gibbs", "ecm"),
  diagnostics = c("classic", "rank"),
  ...
)

precis(object, ...)

## S3 method for class 'rtirt'
precis(object, pars = NULL, digits = 3, ...)

## S3 method for class 'rtirt_precis'
print(x, digits = attr(x, "digits"), ...)

## S3 method for class 'rtirt'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

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

Arguments

`object`	A sampled model, or one fitted by `ecm()`.
`...`	Unused.
`pars`	Optional regular expression selecting parameters.
`method`	`"auto"` (the Gibbs results when the model was sampled, else the ECM fit), `"gibbs"` or `"ecm"`.
`diagnostics`	`"classic"`: R-hat of Gelman & Rubin (1992) and the effective sample size of coda. `"rank"`: the rank-normalized split R-hat and the bulk and tail effective sample sizes of Vehtari, Gelman, Simpson, Carpenter & Bürkner (2021) (package posterior), which also catch chains with different variances or poorly mixing tails; the table then has `ess` (bulk) and `ess_tail`.
`digits`	Decimals to print.
`x`	The result of `precis()`.
`row.names, optional`	Unused.

Value

`estimates()`: a tibble of class rtirt_tbl (printed with 3 decimals; `x$est` and `as.data.frame(x)` keep the full precision). `coef()`: a named vector, printed with 3 decimals.

References

Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science, 7*, 457-472.

Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., & Bürkner, P.-C. (2021). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC. *Bayesian Analysis, 16*, 667-718.

`fit_indices` *Fit indices*

Description

Criteria from the marginal log-likelihood (person parameters integrated out; see `marginal_loglik()`), which compare models with different latent structures: `DIC` (with `pD` = mean deviance - deviance at the posterior means of the item and structural parameters), `WAIC` (`p_waic`) and PSIS-LOO (`LOOIC`, `SE_LOOIC`; needs the 'loo' package). `DIC_complete` is the complete-data DIC of `dic()` (conditional on the person parameters; for comparisons within one model only). Quantile models use the ALD as a working likelihood, so their criteria are not comparable with mean models or across quantile levels. For a model fitted by `ecm()`: the log-likelihood, AIC and BIC.

Usage

```
fit_indices(object, ...)

## S3 method for class 'rtirt'
fit_indices(object, ...)
```

Arguments

`object`	A sampled model, or one fitted by `ecm()`.
`...`	Unused.

`get_bias` *Bias, RMSE and correlation*

Description

Bias, RMSE and correlation

Usage

```
get_bias(est, true)

get_rmse(est, true)

get_corr(est, true)
```

Arguments

est, true	Estimates and true values.

---

gibbs.block_model *Sample a declared model with stacked steps (deprecated)*

Description

Deprecated with block_model(): use the built-in models, gibbs(data, model = ...).

Usage

```
## S3 method for class 'block_model'
gibbs(
  data,
  steps = NULL,
  n_iter = 2000,
  n_burnin = floor(n_iter/2),
  n_chain = 2,
  n_thin = 1,
  seed = 1,
  init = NULL,
  monitor = NULL,
  ...
)

## S3 method for class 'block_model'
algorithm(object, steps = NULL, format = c("unicode", "latex"), ...)

## S3 method for class 'block_model_fit'
algorithm(object, format = c("unicode", "latex"), ...)
```

Arguments

data	A block_model().
steps	A list of steps, e.g. list(step_mala(theta), step_gibbs(a, b)).
n_iter, n_burnin, n_chain, n_thin, seed	As in run_sampler().
init	Optional starting values: a named list (one value or one per person/item). The default starts near the prior means; with two latent variables give data-based values (e.g. standardized sum scores and mean log times), since a chain can settle in another mode (the ability taking over the role of speed); print(fit) warns when the chains disagree.
monitor	Parameters to store; by default all but the person parameters, whose posterior means and SDs are kept instead (fit$mean).

`...`	Unused.
`object`	A `block_model_fit`, or a `block_model` (then give `steps`).
`format`	`"unicode"` (default) or `"latex"` (an `aligned` block).

Details

Runs the steps once per iteration in the order given (see block_steps) with `run_sampler()` (chains, burn-in, storage, summary). Without `steps`, every parameter that has a Gibbs step gets one (in the order of the declarations) and the others get MALA.

Value

A `block_fit` (see `run_sampler()`) with the model and the steps; `algorithm(fit)` lists the steps with their full conditionals.

---

gibbs.input_data *Run the Gibbs sampler*

Description

Samples a built-in model from its full conditionals: `n_chain` independent chains, each from its own random starting values or from an `ecm()` fit.

Usage

```
## S3 method for class 'input_data'
gibbs(
  data,
  model = NULL,
  itemtype = "2pl",
  speed_var = c("free", "fixed"),
  quantile = NULL,
  cov = NULL,
  intercept = FALSE,
  priors = NULL,
  n_iter = 5000,
  n_chain = 4,
  seed = NULL,
  init = "random",
  control = gibbs_control(),
  ...
)

gibbs(data, ...)
```

Arguments

data	Data from input_data().
model	The model, by name: • "irt": 2PL (no response times); with covariates, a latent regression of ability. • "null": joint RT-IRT (van der Linden, 2007), ability and speed correlated. • "latreg": joint RT-IRT with latent regressions of ability and speed on the covariates. • "latent": speed regressed on ability and the covariates (mean or, with quantile, quantile regression). • "cross": cross-relations of ability with each log response time (normal or, with quantile, ALD residuals). • "mixture": the mixture model of Wang and Xu (2015); not available yet.
itemtype	"2pl", "1pl" (discriminations fixed at 1) or "grm" (not available yet); one value, or one per item (all equal for now).
speed_var	"free" (default) estimates the variance of speed; "fixed" sets it to 1 ("null", "latreg", "cross").
quantile	NULL for the mean model, or the quantile level of the ALD model ("latent": of speed given ability and covariates; "cross": of each log response time). A vector fits one model per level (an rtirt_qset). Only "latent" and "cross" have quantile versions; quantile = with another model is an error.
cov	Names of the covariates of data to use (default all); models without covariates ignore them.
intercept	Add an intercept to the latent regressions (default FALSE: the covariates are centred and the item parameters carry the location).
priors	Prior distributions, from rtirt_priors() (default: rtirt_priors()).
n_iter, n_chain	Iterations per chain and chains.
seed	Random seed (NULL: the current RNG state).
init	"random" (default): each chain draws its own starting values; an ecm() fit of the same data and model: start the chains at its estimates and EAP scores, which shortens the burn-in; "ecm": run ecm() first and start there. If the fit is of other data, another model or other settings, the error says what differs. A quantile model starts at an ecm() fit at the same quantile; several levels take init = "ecm" (an ECM fit per level).
control	Further sampler settings, from gibbs_control(), e.g. gibbs(dat, model = "cross", control = gibbs_control(n_burnin = 500)).
...	Not used: any other argument is an error before sampling starts. A setting of gibbs_control() (n_burnin, n_thin, collapse, fit_indices, verbose) gets the control = gibbs_control(...) form in the message, and a misspelled name (niter) the closest argument (n_iter).

Details

```
fit_ecm <- ecm(dat, model = "cross")
fit_gibbs <- gibbs(dat, model = "cross", init = fit_ecm)
```

Value

A fitted model with `post`: `draws` (a `coda::mcmc.list` of the item and structural parameters, all iterations), `mean` (posterior means after burn-in, an `input_para()`), `person` (posterior mean and SD of ability and speed), `loglik` (complete-data log-likelihood per iteration and chain) and `secs`. Accessors: `summary()`, `coef()`, `estimates()`, `scores()`, `reliability()`, `convergence()`, `fit_indices()`, `plot()`.

References

Wang, C., & Xu, G. (2015). A mixture hierarchical model for response times and response accuracy. *British Journal of Mathematical and Statistical Psychology*, 68, 456-477.

Examples

```
cond <- set_cond(n_subj = 300, n_item = 8)
dat <- sim_data(cond, sim_para(cond, "cross"), "cross")
fit <- gibbs(dat, model = "cross", n_iter = 1000, n_chain = 2, seed = 1)
summary(fit)
head(scores(fit))
```

---

input_data *Data container*

Description

Collects the item responses, response times and covariates of one test (mirrors `InputData()` of the Julia package). Each block is either a matrix / data frame or, with `data`, a vector of column names:

Usage

```
input_data(
  resp,
  time = NULL,
  cov = NULL,
  data = NULL,
  id = NULL,
  log_time = NULL
)
```

Arguments

`resp`	Binary (0/1) responses, persons x items.
`time`	Response times in seconds (positive), same dimensions as `resp`; they are log-transformed. Give log times as `log_time` instead.
`cov`	Person covariates, persons x covariates. The latent regressions have no intercept (unless `gibbs(intercept = TRUE)`), so centre (best: standardize) them.

data	Optional data frame from which resp, time, cov and id are taken by name.
id	Person identifiers (a vector, or a column name of data); default 1..N.
log_time	Log response times, instead of time.

Details

```
dat <- input_data(resp = items, time = paste0(items, "_S"), cov = covs, id = "IDSTUD", data = raw)
dat <- input_data(resp = Y, time = RT, cov = X)
```

Item names (the column names of resp) label all output. Missing values are not supported: drop the incomplete persons first.

The data must be in wide format: one row per person, one column per item (for responses and for times). Data with one row per person and item (long format, as exported by PsychoPy or E-Prime) are refused, recognized by a single item or by ids that repeat on most rows; reshape them first, e.g. `tidyr::pivot_wider(long, id_cols = id, names_from = item, values_from = c(correct, rt))` or `stats::reshape(long, idvar = "id", timevar = "item", direction = "wide")`. A few duplicated ids give a warning. Covariates must be numeric: dummy-code categorical ones first, e.g. `model.matrix(~ grade, data)[, -1]` (k - 1 columns for k categories), and centre them.

Responses coded as text or factors ("correct" / "wrong") must be recoded to 0/1 first; the message shows the values found and the recoding. TRUE / FALSE are accepted.

Units of time. Seconds are expected, but the unit only shifts the time intensities: times in milliseconds give every lambda larger by log(1000) = 6.91 (minutes: smaller by log(60) = 4.09), and a, b, rho, the residual variances and the correlations do not change. Report the unit with lambda. The default prior of lambda in gibbs() (N(0, 10^2)) is meant for seconds, so a median time above 1000 gives a warning that suggests milliseconds.

Value

An object of class input_data with Y, kappa = Y - 1/2, T, log_t, X, items and id.

---

input_para *Parameter container*

Description

Mirrors InputPara: a list of the model parameters (theta, a, b, zeta, lambda, sigma2t, nu, beta, rho, b_theta, sigma_p). Used for true values in simulations and for posterior means (fit$post$mean).

Usage

```
input_para(...)
```

Arguments

...	Named parameter values.

julia_aliases *Julia-compatible aliases*

Description

The function names of ExtendedRtIrtModeling.jl, mapped to the snake_case functions:

Usage

```
setCond(
  nSubj = 2000,
  nItem = 15,
  nFeat = 3,
  nIter = 5000,
  nChain = 4,
  nBurnin = NULL,
  nThin = 1,
  nRep = 10,
  qRa = 0.5,
  qRt = 0.5
)

InputData(Y, T = NULL, X = NULL)

InputData4R(Y, kappa = Y - 0.5, logT = NULL, X = NULL)

InputPara(...)

GibbsMlIrt(Cond, Data, truePara = NULL, Para = NULL)

GibbsRtIrtNull(Cond, Data, truePara = NULL, Para = NULL)

GibbsRtIrt(Cond, Data, truePara = NULL, Para = NULL)

GibbsRtIrtLatent(Cond, Data, truePara = NULL, Para = NULL)

GibbsRtIrtLatentQr(Cond, Data, truePara = NULL, Para = NULL)

GibbsRtIrtCross(Cond, Data, truePara = NULL, Para = NULL)

GibbsRtIrtCrossQr(Cond, Data, truePara = NULL, Para = NULL)

sampleGibbs(MCMC, cov2one = NULL, ...)

getDic(MCMC)

checkConvergence(MCMC, ...)
```

```
comparePara(MCMC, par = "a")

getBias(est, true)

getRmse(est, true)

getCorr(est, true)

setTrueParaMlIrt(Cond)

setTrueParaRtIrt(Cond, trueCorr = 0.3)

setTrueParaRtIrtLatent(Cond)

setTrueParaRtIrtCross(Cond)

setDataMlIrt(Cond, truePara)

setDataRtIrt(Cond, truePara)

setDataRtIrtNull(Cond, truePara)

setDataRtIrtLatent(Cond, truePara, type = "norm")

setDataRtIrtCross(Cond, truePara, type = "norm")

runSimulation(
  Cond,
  truePara,
  funcData = setDataRtIrt,
  funcGibbs = GibbsRtIrt,
  par = c("a", "b", "theta"),
  ...
)
```

Arguments

`nSubj`, `nItem`, `nFeat`, `nIter`, `nChain`, `nBurnin`, `nThin`, `nRep`, `qRa`, `qRt`
: As in `setCond()`.

`Y`, `T`, `X`, `kappa`, `logT`
: Data, as in `InputData()` / `InputData4R()`.

`Cond`, `Data`, `truePara`, `Para`, `MCMC`, `par`, `est`, `true`, `...`
: As in the Julia package.

`cov2one`
: Julia's setting: `TRUE` fixes the variance of speed at 1 (`speed_var = "fixed"`), `FALSE` frees it.

`trueCorr`, `type`, `funcData`, `funcGibbs`
: As in the Julia functions.

Details

Julia	birtRcpp
`setCond(nSubj, nItem, nFeat, nIter, nChain, nBurnin, nThin, nRep, qRa, qRt)`	`set_cond()`
`InputData(Y, T, X)`, `InputData4R(Y, kappa, logT, X)`	`input_data()` (resp, time, cov, l
`InputPara(...)`	`input_para()`
`GibbsMlIrt`, `GibbsRtIrtNull`, `GibbsRtIrt`	`mlirt()`, `rtirt_null()`, `rtirt_l`
`GibbsRtIrtLatent`, `GibbsRtIrtLatentQr`	`rtirt_latent()`, `quantile = NUL`
`GibbsRtIrtCross`, `GibbsRtIrtCrossQr`	`rtirt_cross()`, `quantile = NULL`
`sample!(MCMC)`	`sampleGibbs(MCMC) = gibbs()`
`coef(MCMC)`, `precis(MCMC)`	`summary()`, `precis()`
`getDic`, `checkConvergence`, `comparePara`	`dic()`, `convergence()`, `compare_`
`getBias`, `getRmse`, `getCorr`	`get_bias()`, `get_rmse()`, `get_co`
`setTruePara<Model>`, `setData<Model>`	`sim_para()`, `sim_data()`
`runSimulation(Cond, truePara; funcData, funcGibbs)`	`run_simulation()`

The constructors take `(Cond, Data, truePara, Para)` as in Julia and keep the Julia defaults: the speed variance is fixed at 1 (`speed_var = "fixed"`) in `GibbsRtIrtNull`, `GibbsRtIrt` and `GibbsRtIrtCross` (`sampleGibbs(M, cov2one = FALSE)` frees it), and the `Qr` versions use the quantile level `Cond$q_rt`. `setCond()` keeps the Julia default dimensions (2000 persons, 15 items, 3 covariates), which must match the data.

`mala` *Metropolis-adjusted Langevin steps*

Description

A Metropolis–Hastings step whose proposal follows the gradient of the log density (MALA; Roberts & Tweedie, 1996): $x^* = x + \frac{\epsilon^2}{2}\nabla \log \pi(x) + \epsilon z$, accepted with the usual MH probability (the proposal is not symmetric, so its density in both directions enters the ratio). It replaces a random-walk Metropolis step where a block has no conjugate full conditional: it moves in the direction of higher density and needs far fewer rejections, especially in several dimensions (optimal acceptance about 0.574 against 0.234).

Usage

```
draw_mala(x, logpost, step, joint = FALSE)

mcmc_adapt(step, accepted, iter, target = 0.574)
```

Arguments

`x`	Current values: a vector (n scalar targets, or one target with `joint = TRUE`) or an n x d matrix (n independent targets of length d).
`logpost`	Function of `x` returning `list(lp = , grad = )`.

step	Step size(s) ϵ.
joint	FALSE: independent targets, one per row of x; TRUE: x is one target.
accepted	Logical (or acceptance rate) of the last step(s).
iter	Iteration number (for the decreasing adaptation rate).
target	Target acceptance rate.

Details

- draw_mala(x, logpost, step, joint): by default one step for many independent targets at once, one per row of x (or per element of a vector x, e.g. one per person); logpost(x) returns list(lp = , grad =): the log density of every row (a vector) and its gradient (like x); step is one number or one per row. With joint = TRUE, x is one target and lp one number. Returns x with the attribute accepted.
- A parameter on a transformed scale (e.g. u = log a) needs the log Jacobian in lp (+ u) and in grad (+ 1); variables integrated out by the proposal (e.g. Pólya–Gamma omega) must be drawn anew before the next step uses them.
- mcmc_adapt(step, accepted, iter, target): Robbins–Monro adaptation of the step size during burn-in (Andrieu & Thoms, 2008), $\log \epsilon \leftarrow \log \epsilon + t^{-0.6}(\alpha - \alpha^*)$; stop adapting after burn-in so that the chain is a proper Markov chain.

draw_cor() uses it with method = "mala".

Value

draw_mala(): the new x with attribute accepted; mcmc_adapt(): the new step(s).

References

Roberts, G. O., & Tweedie, R. L. (1996). Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli, 2*, 341-363.

Andrieu, C., & Thoms, J. (2008). A tutorial on adaptive MCMC. *Statistics and Computing, 18*, 343-373.

Examples

```
# abilities of a 2PL with known items: one MALA step per person, all persons at once
set.seed(1); N <- 200; a <- c(1, 1.5, 2); b <- c(-1, 0, 1)
theta0 <- rnorm(N); Y <- matrix(rbinom(N * 3, 1, plogis(sweep(outer(theta0, a), 2, a * b))), N)
lp <- function(th) { p <- plogis(sweep(outer(th, a), 2, a * b))
  list(lp = rowSums(Y * log(p) + (1 - Y) * log(1 - p)) - th^2 / 2,
       grad = drop((Y - p) %*% a) - th) }
th <- rep(0, N); eps <- rep(1, N)
for (t in 1:2000) {
  th <- draw_mala(th, lp, eps)
  if (t <= 500) eps <- mcmc_adapt(eps, attr(th, "accepted"), t)
}
mean(attr(th, "accepted"))
```

marginal_loglik *Marginal log-likelihood and fit indices*

Description

`marginal_loglik()` returns the log-likelihood of each person with the person parameters (ability, speed) integrated out, for posterior draws of the item and structural parameters. `fit_indices()` uses it for DIC, WAIC and PSIS-LOO (with the 'loo' package); these, unlike the complete-data DIC of `dic()`, compare models with different latent structures (Merkle, Furr & Rabe-Hesketh, 2019). Ability is integrated by Gauss-Hermite quadrature and speed in closed form (normal log times; normal-ALD convolution for `rtirt_latent` quantile models; piecewise-linear ALD log-likelihood for `rtirt_cross`). Quantile models use the ALD as a working likelihood, so their criteria are not comparable with mean models or across levels.

Usage

```
marginal_loglik(object, n_draws = NULL, Q = NULL)
```

Arguments

`object`	A sampled model.
`n_draws`	Number of posterior draws (after burn-in, evenly spaced; default 200, 100 for `rtirt_cross` quantile models).
`Q`	Gauss-Hermite nodes for ability (default 41; 61 for `rtirt_cross` quantile models, whose integrand in ability has sharper features: about 3e-4 per person from the exact value).

Value

A draws x persons matrix (printed with 3 decimals; see rtirt_tbl).

mml *Direct maximization for a declared model (deprecated)*

Description

Deprecated with `block_model()`: use the built-in models, `ecm(data, model = ...)`.

Usage

```
mml(model, ...)

## S3 method for class 'block_model'
mml(
  model,
  latent = NULL,
  nodes = NULL,
  prior = c("map", "ml"),
  init = NULL,
  cycles = 6,
  max_iter = 1000,
  tol = 1e-09,
  se = TRUE,
  analytic = TRUE,
  quadrature = c("adaptive", "ba81"),
  method = c("newton", "quasi-newton"),
  ...
)
```

Arguments

model	A block_model().
...	Unused.
latent	Parameters integrated out in the E-step (one to three [person] parameters with a normal prior).
nodes	Quadrature points per latent variable (default 21, 11 or 7 for one, two or three); one number, or one per latent variable (named, e.g. c(theta = 11, zeta = 5)).
prior	"map" (posterior mode) or "ml" (maximum marginal likelihood).
init	Optional starting values (named list).
cycles	Maximum number of recentring rounds of the quadrature.
max_iter, tol	Passed to run_em().
se	Standard errors from the Hessian of the objective.
analytic	Integrate latent variables that enter the model only through normal parts, linearly (e.g. speed in a log-normal RT model), exactly given the others instead of on the grid.
quadrature	"adaptive" (default): nodes centred and scaled per person. "ba81": the same nodes for every person (Bock & Aitkin, 1981; default 41, 21 or 11 per latent variable), centred at the marginal posterior. A part of the data that depends on the persons only through the variables on the grid is then summed over persons at each node ("artificial data"), so the E-step is a matrix product and the M-step works on nodes x items: faster for many persons. A latent variable whose posterior is narrow compared with its spread over persons needs many common nodes; the fit compares its objective with adaptive nodes at the estimates (fit$adaptive_gap) and warns when they differ by more than 0.5 (with

`analytic = FALSE` speed is then on the common grid too, and its posterior is narrow when there are many items: use `analytic = TRUE`). Still ECM: only the numerical integration of the E-step changes, the conditional maximizations are the same.

`method` `"newton"` (Hessian by Louis' formula) or `"quasi-newton"`.

Details

Maximizes the objective of `ecm.block_model()` (the marginal log-likelihood, or the log posterior with `prior = "map"`) directly with `stats::nlminb()` (within the bounds of the parameters), all parameters together, with the gradient from Fisher's identity and, for `method = "newton"`, the Hessian from Louis' formula (Glas & van der Linden, 2010, Sec. 3.1, use Newton-Raphson or EM). No closed-form step is needed, only the derivatives of the declared expressions, so this helps when a parameter has no closed-form step. It reaches the same maximum as `ecm()` inside the bounds of the parameters (ECM keeps a bounded parameter 1e-3 inside its bound; a parameter at its bound gets no standard error, with a warning); ECM never lowers the objective and its iterations are cheap, Newton needs few iterations.

Value

As `ecm.block_model()`.

References

Glas, C. A. W., & van der Linden, W. J. (2010). Marginal likelihood inference for a model for item responses and response times. *British Journal of Mathematical and Statistical Psychology, 63*, 603-626.

`models` *Model constructors (deprecated)*

Description

Deprecated: name the model in `ecm()` or `gibbs()` instead, e.g. `rtirt_cross(dat)` becomes `ecm(dat, model = "cross")` or `gibbs(dat, model = "cross")`. The constructors still work (with a warning once per session) and get no new features.

Usage

```
mlirt(
  data,
  itemtype = c("2pl", "1pl"),
  intercept = FALSE,
  cond = NULL,
  true_para = NULL,
  para = NULL
)
```

```
rtirt_null(
  data,
  speed_var = c("free", "fixed"),
  itemtype = c("2pl", "1pl"),
  cond = NULL,
  true_para = NULL,
  para = NULL
)

rtirt_latreg(
  data,
  speed_var = c("free", "fixed"),
  itemtype = c("2pl", "1pl"),
  intercept = FALSE,
  cond = NULL,
  true_para = NULL,
  para = NULL
)

rtirt_latent(
  data,
  quantile = NULL,
  itemtype = c("2pl", "1pl"),
  intercept = FALSE,
  cond = NULL,
  true_para = NULL,
  para = NULL
)

rtirt_cross(
  data,
  quantile = NULL,
  speed_var = c("free", "fixed"),
  itemtype = c("2pl", "1pl"),
  cond = NULL,
  true_para = NULL,
  para = NULL
)
```

Arguments

`data`	Data from `input_data()`.
`itemtype`	`"2pl"` or `"1pl"` (discriminations fixed at 1).
`intercept`	Add an intercept to the latent regressions (default `FALSE`: the covariates are centred and the item parameters carry the location).
`cond`	Optional `set_cond()` (simulations): the dimensions are checked against the data, and its MCMC settings are the defaults of `gibbs()`.

`true_para`	Optional true parameters (`input_para()`) for simulations; by default those stored by `sim_data()`.
`para`	Optional initial values (not needed; each chain draws its own).
`speed_var`	`"free"` (default) estimates the variance of speed; `"fixed"` sets it to 1 (`rtirt_latreg`, `rtirt_null`: ability and speed then have a correlation matrix), as the Julia package does. For log seconds the speed variance is usually far below 1, so fixing it over-constrains the model.
`quantile`	`NULL` for the mean model, or the quantile level of the ALD model (`rtirt_latent`: of speed given ability and covariates; `rtirt_cross`: of each log response time). A vector fits one model per level (an `rtirt_qset`; `gibbs()` samples all, `plot()` draws the coefficients against the level).

Details

```
fit <- gibbs(dat, model = "cross", quantile = 0.25, n_iter = 10000, n_chain = 3, seed = 1)
```

- `mlirt`: 2PL with a latent regression of ability (no response times).
- `rtirt_null`: joint RT-IRT (van der Linden, 2007), ability and speed correlated.
- `rtirt_latreg`: joint RT-IRT with latent regressions of ability and speed on covariates (dissertation Chapter 3).
- `rtirt_latent`: speed regressed on ability and covariates, mean regression or, with `quantile`, quantile regression (ALD; Chapter 4).
- `rtirt_cross`: cross-relations of ability with each log response time, normal or, with `quantile`, ALD residuals (Chapter 5).

The Julia constructors (`GibbsRtIrtCross(Cond, Data)` etc.) are aliases with the Julia argument order and defaults (see julia_aliases).

Value

A model object of class `c(<model>, "rtirt")`; see `gibbs()`.

`normal_parts` *Normal parts of full conditionals*

Description

Most full conditionals of the samplers are normal. A *normal part* stores one as its precision Q and its linear term h (`prec` and `num`; mean $Q^{-1}h$, variance Q^{-1}), so that the contributions of the data and of the prior simply add up:

Usage

```
normal_part(prec, num)

normal_prior(mean = 0, sd = 1)

reg_part(X, y, w = 1)

## S3 method for class 'normal_part'
e1 + e2

draw(part, lower = -Inf, upper = Inf)

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

Arguments

prec, num	Precision (a vector, or a matrix for a regression part) and linear term.
mean, sd	Prior mean and SD (recycled).
X	Design matrix (one row per observation).
y	Observations.
w	Precisions of the observations (one value or one per observation).
e1, e2	Normal parts.
part, x	A normal part.
lower, upper	Truncation bounds.
...	Unused.

Details

- irt_part() and calc_normal_part() return the data part of independent scalars (one per person or item);
- reg_part(X, y, w) returns the data part of a vector of regression coefficients g in $y = Xg+e$, $e_j \sim N(0, 1/w_j)$ ($Q = X'WX$, $h = X'Wy$), e.g. a latent regression of the abilities on covariates;
- normal_prior(mean, sd) turns a normal prior into a part ($Q = 1/sd^2$, $h = mean/sd^2$);
- part1 + part2 adds parts (elementwise, or a prior added to the diagonal of a regression part);
- draw(part, lower, upper) draws from it (truncated to [lower, upper] when given; a regression part is drawn with draw_mvn());
- print(part) shows the conditional means and SDs.

```
theta <- draw(irt_part(Y, omega, a, b) + normal_prior(mean = X %*% g, sd = 1))
g <- draw(reg_part(X, theta) + normal_prior(0, 10))
```

Value

normal_prior(), normal_part(), reg_part() and +: a normal part; draw(): a vector of draws.

Examples

```
set.seed(1)
N <- 200; a <- c(1, 1.5, 2); b <- c(-1, 0, 1); th <- rnorm(N)
Y <- matrix(rbinom(N * 3, 1, plogis(sweep(outer(th, a), 2, a * b))), N)
om <- draw_omega(a, b, th)
p <- irt_part(Y, om, a, b) + normal_prior(0, 1)        # data part + prior N(0, 1)
p
theta <- draw(p)
draw(normal_prior(1, 1), lower = 0)                     # N+(1, 1)
X <- cbind(1, rnorm(N))
draw(reg_part(X, theta) + normal_prior(0, 10))          # coefficients of theta on X
```

`person_fit` *Person fit of the response times*

Description

The person-fit statistic of Marianti, Fox, Avetisyan, Veldkamp & Tijmstra (2014) for the lognormal RT model: $l_j^t = \sum_i (\log t_{ij} - \lambda_i + \zeta_j[+\rho_i\theta_j])^2/\sigma_i^2$, chi-square with K degrees of freedom when the model holds. The samplers of `gibbs()` evaluate it in every iteration after burn-in, so the result accounts for the uncertainty of all parameters:

- `lt`: posterior mean of l_j^t;
- `p_flag`: posterior probability that l_j^t exceeds the 95% quantile of chi-square(K), the probability that the pattern is flagged; Marianti et al. flag a pattern when it is at least .95.

Usage

```
person_fit(object, flag = 0.95, ...)
```

Arguments

`object`	A model sampled by `gibbs()`.
`flag`	Posterior probability above which a pattern is flagged.
`...`	Unused.

Details

Large values point to aberrant timing: rapid guessing, item preknowledge, running out of time, or a speed that changes during the test. Normal log times only (not `gibbs(model = "cross", quantile = )`).

Value

A data frame (class rtirt_tbl, printed with 3 decimals) with `id`, `lt`, `df`, `p_flag` and `flagged`; attribute `share` (share of flagged test takers) and `share_mean` (share with `lt` above the 95% quantile).

References

Marianti, S., Fox, J.-P., Avetisyan, M., Veldkamp, B. P., & Tijmstra, J. (2014). Testing for aberrant behavior in response time modeling. *Journal of Educational and Behavioral Statistics, 39*, 426-451.

Fox, J.-P., & Marianti, S. (2017). Person-fit statistics for joint models for accuracy and speed. *Journal of Educational Measurement, 54*, 243-262.

Examples

```
cond <- set_cond(n_subj = 300, n_item = 10)
d <- sim_data(cond, sim_para(cond, "cross"), "cross")
f <- gibbs(d,
  model = "null", n_iter = 2000, n_chain = 2, seed = 1,
  control = gibbs_control(verbose = FALSE, fit_indices = FALSE)
)
pf <- person_fit(f)
attr(pf, "share") # close to 0 under the model
```

---

plot.rtirt *Plots of a sampled model*

Description

`type = "items"`: posterior means and 95% intervals of the item parameters (ECM fits: estimates and Wald 95% intervals), one panel per parameter type (a, b, lambda, sigma2t, rho). `type = "structural"`: the same for the structural parameters (regression coefficients, `cor_ability_speed`, `b_ability`, `var_speed`), or those matching `pars`. `type = "trace"`: trace plots (all iterations; burn-in shaded) of the structural parameters or of those matching `pars`. For plots of your own (e.g. with ggplot2), `estimates(fit)` (or `as.data.frame(fit)`) has the estimates and intervals as a data frame.

Usage

```
## S3 method for class 'rtirt'
plot(x, type = c("items", "structural", "trace"), pars = NULL, ...)
```

Arguments

`x`	A sampled model.
`type`	`"items"`, `"structural"` or `"trace"`.
`pars`	Regular expression selecting parameters (`"structural"` and `"trace"`; at most 12 are drawn in `"trace"`).
`...`	Passed to `graphics::plot()`.

`preknowledge_test` *Tests of item preknowledge*

Description

Statistics of Sinharay (2020) for test takers who may have known some items in advance (a set of possibly compromised items is given), computed from the item parameters of a fit of a model with response times (`ecm()` estimates, or the posterior means of `gibbs()`), with the lognormal RT model $\log t_{ij} \sim N(\lambda_i - \tau_j, \sigma_i^2)$ and the 2PL for the scores:

- `chi_pf`: RT person fit $\sum_i (\log t_{ij} - \lambda_i + \hat{\tau}_j)^2/\sigma_i^2$ with the speed estimated from the times, chi-square with K - 1 df (it does not use the compromised set);
- `Lambda_s`: signed likelihood ratio (a z statistic) comparing the speed on the compromised items with the speed on the other items, $(\hat{\tau}_c - \hat{\tau}_{\bar{c}})/\sqrt{1/W_c + 1/W_{\bar{c}}}$, $W = \sum 1/\sigma_i^2$; large values: faster on the compromised items;
- `L_s`: signed likelihood ratio of the item scores comparing the ability (maximum likelihood) on the two sets; large values: better on the compromised items. The p-values are one-sided for `Lambda_s` and `L_s` (standard normal) and upper-tail for `chi_pf`. Models with the speed entering as $\lambda_i - \zeta_j$: `model = "null"`, `"latreg"` and `"latent"` (mean model).

Usage

```
preknowledge_test(object, compromised, ...)
```

Arguments

`object`	A model fitted by `ecm()` or `gibbs()`.
`compromised`	Items in the compromised set (names or a logical / integer index).
`...`	Unused.

Value

A data frame (class rtirt_tbl, printed with 3 decimals) with one row per test taker: the three statistics and their p-values.

References

Sinharay, S. (2020). Detection of item preknowledge using response times. *Applied Psychological Measurement, 44*, 376-392.

Examples

```
cond <- set_cond(n_subj = 400, n_item = 20)
d <- sim_data(cond, sim_para(cond, "null"), "null")
pk <- preknowledge_test(ecm(d, model = "null", se = FALSE), compromised = 1:5)
colMeans(pk[c("p_chi_pf", "p_Lambda_s", "p_L_s")] < 0.01)    # about .01 under the model
```

reliability *Empirical reliability*

Description

var(EAP) / (var(EAP) + mean PSD^2) of ability and speed.

Usage

```
reliability(object, ...)

## S3 method for class 'rtirt'
reliability(object, ...)
```

Arguments

`object`	A sampled model.
`...`	Unused.

rtirt_priors *Priors of the Gibbs samplers*

Description

Sets the prior distributions used by `gibbs()` (and, for the item parameters, by `ecm(prior = "map")`). The defaults are common weakly informative priors that do not depend on the data, and every full conditional of the samplers stays conjugate (pure Gibbs):

Usage

```
rtirt_priors(
  a = c(mean = 1, sd = 1),
  b = c(mean = 0, sd = 3),
  lambda = c(mean = 0, sd = 10),
  sigma2t = c(df = 3, scale = 1),
  beta = c(sd = 1),
  rho = c(sd = 1),
  var_speed = c(df = 3, scale = 1),
  cov = c(sd = 1)
)

rtirt_priors_julia()
```

Arguments

a	Discrimination, normal on (0, Inf): c(mean, sd).
b	Difficulty, normal: c(mean, sd) or c(mean, sd, lower, upper) (truncated).
lambda	Time intensity, normal: c(mean, sd) or c(mean, sd, lower) (truncated below). Only rtirt_priors_julia() centres it on the data (mean and SD of all log response times, truncated at 0), to reproduce ExtendedRtIrtModeling.jl; every other prior is fixed in advance.
sigma2t	Residual variance (or ALD scale) of the log response times: c(df = , scale =) (half-t on its square root) or c(shape = , scale =) (inverse gamma).
beta	Regression coefficients of the latent regressions (including b_ability of "latent"), normal with mean 0: c(sd).
rho	Cross-relations of model = "cross", normal with mean 0: c(sd).
var_speed	Speed variance ("cross"), residual speed variance ("latreg", "null") or its ALD scale ("latent"): c(df = , scale =) or c(shape = , scale =), as sigma2t.
cov	Covariance c of speed with ability in "latreg" / "null" with free speed variance, normal with mean 0: c(sd) (with speed_var = "fixed" the correlation is uniform on (-1, 1)).

Details

parameter	default	Gibbs step
a (discrimination)	N(1, 1) on (0, Inf)	truncated normal
b (difficulty)	N(0, 3^2), not truncated	normal
lambda (time intensity)	N(0, 10^2), not truncated	normal
sigma2t (residual variance of log RT)	its SD ~ half-t(3, 1)	two inverse-gamma steps
speed (residual) variance	its SD ~ half-t(3, 1)	two inverse-gamma steps
regression coefficients, rho, c	N(0, 1)	normal

The half-t prior on a standard deviation (Gelman, 2006) is the scale mixture of Huang & Wand (2013): var | u ~ IG(df / 2, df / u), u ~ IG(1 / 2, 1 / scale^2). The samplers draw u | var ~ IG((df + 1) / 2, df / var + 1 / scale^2) and then var | u, data ~ IG(df / 2 + n / 2, df / u + SS / 2), both conjugate. The auxiliary u is drawn afresh from the current variance in every iteration, so it is not stored.

rtirt_priors_julia() gives the priors of ExtendedRtIrtModeling.jl (Tsai, 2025), with which the dissertation results are reproduced: a ~ N+(1, 1), b ~ N(0, 1) on [-4, 4], lambda ~ N+ centred at the mean and SD of the log response times, inverse-gamma(0.001, 0.001) variances.

A variance with c(df, scale) has a half-t prior on its square root and with c(shape, scale) an inverse-gamma prior on the variance (one conjugate step).

Value

An object of class rtirt_priors.

References

Gelman, A. (2006). Prior distributions for variance parameters in hierarchical models. *Bayesian Analysis, 1*, 515-534.

Huang, A., & Wand, M. P. (2013). Simple marginally noninformative prior distributions for covariance matrices. *Bayesian Analysis, 8*, 439-452.

Examples

```
rtirt_priors()                                  # the defaults
rtirt_priors(a = c(mean = 1, sd = 0.5))         # a tighter discrimination prior
rtirt_priors_julia()                            # the priors of the dissertation
```

`rtirt_qset` *Several quantile levels*

Description

`fits <- gibbs(data, model = "cross", quantile = c(.1, .5, .9))` (or `model = "latent"`) samples one ALD model per level and returns them as an `rtirt_qset`; `ecm(data, model = "cross", quantile = c(.1, .5, .9))` fits them by maximum (working) likelihood. `coef()` gives a parameters x levels matrix, `estimates()` stacks the tables with a column `q`, and `plot()` draws the quantile coefficients (cross-relations `rho`, or the regression coefficients of `"latent"`) against the level with 95% intervals (posterior intervals, or Wald intervals from the sandwich standard errors of `ecm()`). `convergence()` checks every level. `summary()` returns an object that prints as `print()`. The fit of one level is `fits$models[[k]]` (an ordinary fit: `summary()`, `estimates()`, `plot(type = "trace")` for a sampled fit).

Usage

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

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

## S3 method for class 'rtirt_qset'
estimates(object, ...)

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

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

## S3 method for class 'rtirt_qset'
plot(x, pars = NULL, ...)
```

Arguments

x, object	An rtirt_qset.
...	Unused, or passed to graphics::plot().
pars	Regular expression of the parameters to plot (default: rho for "cross", beta and b_ability for "latent").

---

rtirt_summary *Summary tables*

Description

summary() of a sampled model (or one fitted by ecm()) returns its tables as one object: header, fit (fit indices), items and se (one row per item: a, b, lambda, sigma2t, rho and their posterior SDs or standard errors), parameters (one row per parameter with its interval and, for Gibbs fits, its prior, as in blavaan), covariance (person covariance) and reliability. Each element is a tibble; as.data.frame() returns parameters. print() shows the tables under their names, so s$items gets the one printed as $items.

Usage

```
## S3 method for class 'rtirt_summary'
print(x, tables = NULL, digits = 3, ...)

## S3 method for class 'rtirt_summary'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'rtirt_table'
print(x, digits = 3, title = TRUE, ...)
```

Arguments

x	An rtirt_summary, or one of its tables.
tables	Names of the tables to print (default all); an unknown name is an error.
digits	Decimals.
...	Unused.
row.names, optional	Unused.
title	Print the title of a table.

Value

print() returns x invisibly; as.data.frame() the parameter table.

rtirt_tbl *Printing the tables of birtRcpp*

Description

The tables of `estimates()`, `scores()`, `fit_indices()`, `compare_para()`, `person_fit()`, `preknowledge_test()`, `dic()` and `run_simulation()` are tibbles (or data frames) of class `rtirt_tbl`. They print numbers with 3 decimals (numbers of 1e10 or more, such as the bound of a diverging estimate, as e.g. `5.950e+99`; counts without decimals, p-values below 0.001 as `<0.001`, `NA` blank), and all rows of a table with up to 40 rows (else the first 20). The values are not rounded: `x$est`, dplyr verbs and `as.data.frame(x)` have the full precision. Named vectors and matrices of numbers (coef(), `reliability()`, `marginal_loglik()`) print with 3 decimals as well.

Usage

```
## S3 method for class 'rtirt_tbl'
format(x, digits = 3, n = NULL, ...)

## S3 method for class 'rtirt_tbl'
print(x, digits = 3, n = NULL, ...)

## S3 method for class 'rtirt_values'
print(x, digits = 3, ...)
```

Arguments

`x`	A table of birtRcpp.
`digits`	Decimals.
`n`	Rows to show (default: all up to 40 rows, else 20); `Inf` for all.
`...`	Unused.

Value

`format()` the lines; `print()` returns `x` invisibly.

Examples

```
cond <- set_cond(n_subj = 200, n_item = 5)
fit <- ecm(sim_data(cond, sim_para(cond, "null"), "null"), model = "null", se = FALSE)
est <- estimates(fit)
est            # 3 decimals
est$est[1]     # full precision
as.data.frame(est)[1:2, ]
```

run_em *EM algorithms from components, with the bookkeeping done*

Description

The EM counterpart of `run_sampler()`: the algorithm is one function `step(s, data)` that does one E- and M-step from the current state `s` (a named list of parameters) and returns the new state with the log-likelihood **at the input state** in `s$.loglik`. `run_em()` packs the state into a vector (log scale for the `positive` parameters), accelerates with SQUAREM, checks convergence and computes standard errors from the numerical Hessian of `loglik()`.

Usage

```
run_em(
  data,
  init,
  step,
  loglik = NULL,
  positive = character(),
  se = !is.null(loglik),
  max_iter = 1000,
  tol = 1e-10,
  accelerate = TRUE,
  score = NULL,
  hessian = NULL
)

## S3 method for class 'block_em'
print(x, digits = 3, ...)

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

estep_irt(Y, a, b, mean = 0, sd = 1, nodes = calc_nodes(41))

mstep_items(e, a, b, priors = NULL, b_mean = NULL, one_pl = FALSE)
```

Arguments

`data`	A list of data.
`init`	Function of `data` returning the starting state (or the state).
`step`	Function of `(s, data)`: one EM step; returns the new state with `.loglik`.
`loglik`	Function of `(s, data)` giving the marginal log-likelihood (plus the log prior for posterior modes), for the standard errors and the checks; `NULL`: neither.
`positive`	Names of the parameters kept positive (estimated on the log scale).

se	Compute standard errors.
max_iter, tol, accelerate	As em_squarem().
score	Optional function of (s, data) giving the gradient of loglik() as a named list (one vector per parameter, natural scale), e.g. from Fisher's identity; the Hessian for the standard errors is then its numerical derivative (2p calls instead of about 4p^2 calls of loglik()).
hessian	Optional function of (s, data) giving the Hessian of loglik() on the natural scale (a matrix, parameters in the order of the state), e.g. from Louis' formula; used instead of numerical differentiation (NULL result: numerical).
digits	Number of decimals.
...	Unused.
object, x	A block_em.
Y	Responses (0/1, NA missing), N x K.
a, b	Item parameters.
mean, sd	Mean (one value or one per person) and SD of the ability distribution.
nodes	Nodes from calc_nodes().
e	Result of estep_irt().
priors	Item priors from rtirt_priors() (posterior mode); NULL: maximum likelihood.
b_mean	Prior means of b, one per item (NULL: the mean of priors$b).
one_pl	Keep a fixed.

Details

For posterior modes (priors in the M-step), s$.loglik and loglik() are the log-likelihood plus the log prior: the objective that the M-step increases.

run_em() checks the step, because a wrong M-step still converges, to the wrong values. It warns when the objective decreases between iterations (an EM step never lowers it), when s$.loglik differs from loglik() at the starting values, and when the estimates are not a maximum of loglik() (score statistic $g'(-H)^{-1}g > 0.01$).

For item response models the E- and M-steps are ready-made:

- estep_irt(Y, a, b, mean, sd, nodes): the Gauss-Hermite E-step of a 2PL with $\theta_j \sim N(\text{mean}_j, \text{sd}^2)$ (missing responses skipped): posterior weights of the nodes W, the marginal log-likelihood loglik, the EAPs and posterior SDs, and the Pólya–Gamma sums for the M-step.
- mstep_items(e, a, b, priors, b_mean, one_pl): the item parameters from the E-step (Pólya–Gamma minorizer: never lowers the objective); maximum likelihood, or the posterior mode with the item priors of priors = rtirt_priors(a = , b =) and, optionally, a prior mean of b per item (b_mean), as draw_items_pg().

The components behind them are in em_components.

Value

`run_em()`: a `block_em` with `estimates` (estimate and SE), `state`, `loglik`, `iterations`, `converged`, `trace` and `vcov` (working scale). `estep_irt()`: a list; `mstep_items()`: `list(a, b)`.

Examples

```
set.seed(1)
N <- 1000; K <- 8; a0 <- runif(K, 0.8, 2); b0 <- rnorm(K)
Y <- matrix(rbinom(N * K, 1, plogis(sweep(outer(rnorm(N), a0), 2, a0 * b0))), N)
fit <- run_em(
  data = list(Y = Y),
  init = function(d) list(a = rep(1, ncol(d$Y)), b = rep(0, ncol(d$Y))),
  step = function(s, d) {
    e <- estep_irt(d$Y, s$a, s$b)                # E-step
    s[c("a", "b")] <- mstep_items(e, s$a, s$b)   # M-step
    s$.loglik <- e$loglik
    s
  },
  loglik = function(s, d) estep_irt(d$Y, s$a, s$b)$loglik,
  positive = "a")
fit
```

---

run_sampler *Run a sampler written from the building blocks*

Description

The loop around a sampler: chains, burn-in, thinning, storage, running means of person parameters and the summary table, as in JAGS. The sampler is one function `step(s, data)` that takes the current state `s` (a named list of everything the sampler updates) and returns the new state, as in a JAGS model block written out by hand.

Usage

```
run_sampler(
  data,
  init,
  step,
  monitor,
  n_iter = 2000,
  n_burnin = floor(n_iter/2),
  n_chain = 2,
  n_thin = 1,
  seed = 1,
  keep_mean = NULL
)

## S3 method for class 'block_fit'
```

```
summary(object, ...)

## S3 method for class 'block_fit'
print(x, digits = 3, ...)

## S3 method for class 'block_fit'
as.matrix(x, ...)
```

Arguments

data	A list of data (passed to `init()` and `step()`).
init	Function of `data` returning the starting state (or a list, used by every chain).
step	Function of `(s, data)` returning the new state.
monitor	Names of the elements of the state to store.
n_iter, n_burnin, n_thin	Iterations per chain, burn-in and thinning (as `gibbs()`).
n_chain	Number of chains.
seed	Seed of the first chain (chain c uses `seed + c - 1`).
keep_mean	Names of elements whose posterior mean and SD are kept.
object, x	A `block_fit`.
...	Unused.
digits	Number of decimals.

Details

- `init(data)` returns the starting state (called once per chain, after `set.seed()`, so random starting values differ between chains).
- `step(s, data)` updates every block once and returns `s`. During burn-in `s$.burn` is `TRUE` and `s$.iter` is the iteration, for step-size adaptation (`mcmc_adapt()`).
- `monitor`: the names of the elements of `s` to store (vectors are stored as `a[1]`, `a[2]`, ...).
- `keep_mean`: the names of elements (e.g. person parameters) of which only the posterior mean and SD are kept.

Value

A `block_fit`: `draws` (a `coda::mcmc.list`), `mean` (posterior means and SDs of `keep_mean`), `last` (the last state of every chain) and `time`. `summary()` gives the table of posterior means, SDs, 95% intervals, R-hat and effective sample sizes; `as.matrix()` the draws. `print()` warns when an R-hat exceeds 1.05 or an effective sample size is below 100, and names the usual cures.

See Also

`check_sampler()` to check that the sampler draws from the right posterior, `run_em()`.

Examples

```
set.seed(1)
N <- 300; K <- 6; a0 <- runif(K, 0.8, 2); b0 <- rnorm(K)
Y <- matrix(rbinom(N * K, 1, plogis(sweep(outer(rnorm(N), a0), 2, a0 * b0))), N)
fit <- run_sampler(
  data = list(Y = Y),
 init = function(d) list(theta = rnorm(nrow(d$Y)), a = rep(1, ncol(d$Y)), b = rep(0, ncol(d$Y))),
  step = function(s, d) {
    s$omega <- draw_omega(s$a, s$b, s$theta)                    # Polya-Gamma
    s[c("a", "b")] <- draw_items_pg(d$Y, s$omega, s$theta, s$a)  # b, then a
    s$theta <- draw(irt_part(d$Y, s$omega, s$a, s$b) + normal_prior(0, 1))
    s
  },
  monitor = c("a", "b"), keep_mean = "theta", n_iter = 1000)
fit
```

---

run_simulation *Simulation study*

Description

As runSimulation() in Julia: cond$n_rep replications with fixed item parameters and new persons (and data) in each replication; returns bias, RMSE and correlation per parameter block and replication.

Usage

```
run_simulation(
  cond,
  para,
  model = sim_models,
  quantile = NULL,
  pars = c("a", "b", "ability"),
  type = "norm",
  speed_var = c("free", "fixed"),
  ...
)
```

Arguments

cond	Conditions.
para	True parameters from sim_para().
model	Model name as in sim_para().
quantile	Quantile level of the ALD version ("cross", "latent"), or NULL.
pars	Parameter blocks to evaluate.
type	Residual distribution of the simulated data.
speed_var	Variance of speed ("cross", "latreg", "null"), as in the constructors.
...	Passed to gibbs().

Value

A tibble (class rtirt_tbl, printed with 3 decimals) with one row per replication and parameter block.

`score_test` *Score-based tests of parameter invariance*

Description

Tests whether item or structural parameters change along a person-level moderator (DIF, measurement invariance, item position effects) from a single `ecm()` fit: the casewise scores (`estfun.rtirt()`) are ordered by the moderator, cumulated after decorrelation, and compared with a Brownian bridge (Merkle & Zeileis, 2013; for IRT, Schneider, Strobl, Zeileis & Debelak, 2022).

Usage

```
score_test(object, ...)

## Default S3 method:
score_test(object, ...)

## S3 method for class 'rtirt'
score_test(
  object,
  moderator,
  parm = "accuracy",
  by_item = FALSE,
  test = c("auto", "DM", "CvM", "maxLM", "LM"),
  decorrelate = c("block", "full"),
  cluster = NULL,
  impact_check = TRUE,
  ...
)
```

Arguments

`object`	A model fitted by `ecm()` (fitted first when it has no ECM fit).
`...`	Arguments of the method (`moderator`, `parm`, ...).
`moderator`	A vector with one value per person: numeric (continuous or ordered) or a factor.
`parm`	Parameters to test: `"accuracy"` (a and b of every item, default), `"rt"` (lambda, sigma2t and rho), `"structural"`, a regular expression on the parameter names, or column indices.
`by_item`	`TRUE`: test the parameters of each item separately (a and b, or the RT parameters), with Holm-adjusted p-values, in addition to the joint test.
`test`	`"auto"` (the double-maximum test `"DM"` for numeric moderators, the Lagrange multiplier test `"LM"` for factors), `"DM"`, `"CvM"` (Cramér-von Mises), `"maxLM"` (supremum LM, 10% trimming) or `"LM"`.

`decorrelate`	`"block"` (within the tested parameters) or `"full"` (all parameters, then the tested components are selected; not valid for moderators built from the response times).
`cluster`	Optional cluster identifiers (see Details).
`impact_check`	Message when the moderator is related to the ability EAPs but is not a covariate of the model (see Details).

Details

Moderators derived from the response times. The marginal distribution of the response times does not involve the accuracy parameters (a, b), so their scores have mean zero given any function of the times (for example the total log time): the accuracy parameters can be tested along such a moderator, provided the scores are decorrelated within the tested block (`decorrelate = "block"`, the default). The response time parameters cannot be tested along a moderator built from the times. Speed scores (EAPs) also use the responses and are not such a function.

DIF and impact. The model identifies ability as N(x'beta, 1) for everyone, so a moderator related to ability (impact) shifts all difficulties and is flagged as instability. To test DIF beyond impact, include the moderator as a covariate of the latent regression (`ecm(input_data(..., cov = z), model = "latreg")`, or `model = "irt"` without response times) and test that fit; `score_test()` gives a message when the moderator correlates with the ability EAPs and is not among the covariates.

Clustered data. With `cluster`, the scores are summed within clusters, which is exact for a moderator that is constant within clusters (for example a school characteristic); a person-level moderator with clustered persons is refused.

Value

A data frame of class `rtirt_score_test`: `parameters`, `statistic`, `p.value` and, with `by_item`, `p.holm`.

Examples

```
cond <- set_cond(n_subj = 400, n_item = 8)
dat <- sim_data(cond, sim_para(cond, "cross"), "cross")
fit <- ecm(dat, model = "cross", se = FALSE)
score_test(fit, rbinom(400, 1, 0.5) == 1, by_item = TRUE)     # a random grouping: no DIF
score_test(fit, rowSums(dat$log_t))                            # along the total log time
```

`score_test.block_model_ecm`

Lagrange multiplier test for a declared model

Description

Tests whether a term added to a part of a declared model is needed, from the `ecm()` fit of the model without it (the null model): `score_test(fit, add = Y ~ delta[item] * z)` tests $\delta_i = 0$ in logit $P(Y_{ji} = 1) = \eta_{ji} + \delta_i z_j$, that is uniform DIF along z, for all items together and for each item. Only the null model is fitted.

Usage

```
## S3 method for class 'block_model_ecm'
score_test(object, add, data = NULL, ...)
```

Arguments

`object`	A fit of `ecm()` (or `mml()`) of a `block_model()`: the null model.
`add`	A formula `Y ~ term` (see Details).
`data`	Optional named list (or data frame) of person-level or persons x items data used in the term and not in the model.
`...`	Unused.

Details

The term. `add` is a formula `Y ~ term`: `Y` names an observed part of the model, and `term` is added to its linear predictor (`bernoulli_logit()`) or to its mean (`normal()`). New parameters are written with their dimension as in `block_model()` (`delta[item]`, `delta[person]`) or without one for a single value (`Y ~ delta * z`: the same shift for every item). The term must be 0 when the new parameters are 0, e.g. `delta[item] * z`, or `delta[item] * z * theta` for a change of the discrimination (non-uniform DIF). Other names in the term are parameters of the model or data:

- data of the model (`block_model(Y = Y, z = z, ...)`: a vector with one value per person, or a persons x items matrix, e.g. `logT ~ delta[item] * Y` for the dependence of the times on the responses of van der Linden & Glas, 2010; see `ci_test()`);
- or a vector given in `data = list(z = z)`, with the same rules (no missing values).

The statistic (Rao, 1948; for IRT, Glas, 1998, 1999; Glas & van der Linden, 2010). With s the score of the new parameters δ at the null estimates $\hat{\phi}$ (with $\delta = 0$) and I the observed information of (ϕ, δ),

$$LM = s' W^{-1} s, \quad W = I_{\delta\delta} - I_{\delta\phi} I_{\phi\phi}^{-1} I_{\phi\delta},$$

chi-square with as many degrees of freedom as new parameters when the null model holds; it equals the likelihood ratio statistic asymptotically, without fitting the alternative. W is the information about δ left after the null parameters are estimated; the score of the null parameters is 0 at their estimates. When a direction of the new parameters is absorbed exactly by the null parameters (e.g. a shift of every item and a free mean of the latent variable), its casewise scores are 0 for every person; the joint test is then made in the other directions (found from the cross-product of the casewise scores) and `df` is their number (the per-element tests are not affected).

- The score is the expected complete-data score under the E-step of the fit (Fisher's identity; Louis, 1982), as for the standard errors of `ecm()`.

- The information is the observed information by Louis' formula (expected complete-data Hessian minus the covariance of the complete-data score), on the same quadrature grid. Falk & Monroe (2018) found that LM tests in IRT with the observed (Hessian) information keep their size, and those with the cross-product of the casewise scores reject too often.
- Per element: when the new parameters have one dimension (e.g. `delta[item]`), each item is tested alone (the alternative in which only that item's parameters are free): $s_i' W_{ii}^{-1} s_i$, chi-square with one degree of freedom per new parameter, with Holm-adjusted p-values (Holm, 1979). `estimate` is the one-step estimate $W_{ii}^{-1} s_i$ of that element (the expected parameter change; Sörbom, 1989), shown for one new parameter.

With `prior = "map"` the score and the information are those of the log posterior of the null parameters; the new parameters get no prior. The usual test is from `prior = "ml"`.

DIF and impact. As for the score tests of the built-in models (`score_test.rtirt()`): when z is related to the latent variable, a model whose latent mean does not depend on z shows the difference in mean as a shift of every item. Put z in the prior of the latent variable, `theta[person] ~ normal(g * z, 1)`, to test DIF beyond impact; the test of all items together then has one degree of freedom less (the shift that g absorbs).

Value

A data frame of class `block_lm_test`: one row for the joint test (`all`) and, when the new parameters have one dimension, one row per element (`statistic`, `df`, `p.value`, `p.holm`, `estimate`). Attributes: `add`, `score` and `W`.

References

Falk, C. F., & Monroe, S. (2018). On Lagrange multiplier tests in multidimensional item response theory: Information matrices and model misspecification. *Educational and Psychological Measurement, 78*(4), 653–678. doi:10.1177/0013164417714506

Glas, C. A. W. (1998). Detection of differential item functioning using Lagrange multiplier tests. *Statistica Sinica, 8*(3), 647–667.

Glas, C. A. W. (1999). Modification indices for the 2-PL and the nominal response model. *Psychometrika, 64*(3), 273–294. doi:10.1007/BF02294296

Glas, C. A. W., & van der Linden, W. J. (2010). Marginal likelihood inference for a model for item responses and response times. *British Journal of Mathematical and Statistical Psychology, 63*(3), 603–626. doi:10.1348/000711009X481360

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics, 6*(2), 65–70.

Louis, T. A. (1982). Finding the observed information matrix when using the EM algorithm. *Journal of the Royal Statistical Society B, 44*(2), 226–233. doi:10.1111/j.25176161.1982.tb01203.x

Rao, C. R. (1948). Large sample tests of statistical hypotheses concerning several parameters with applications to problems of estimation. *Mathematical Proceedings of the Cambridge Philosophical Society, 44*(1), 50–57. doi:10.1017/S0305004100023987

Sörbom, D. (1989). Model modification. *Psychometrika, 54*(3), 371–384. doi:10.1007/BF02294623

van der Linden, W. J., & Glas, C. A. W. (2010). Statistical tests of conditional independence between responses and/or response times on test items. *Psychometrika, 75*(1), 120–139. doi:10.1007/s1133600991299

Examples

```
set.seed(1); N <- 400; K <- 6
z <- rbinom(N, 1, 0.5); th <- rnorm(N); b0 <- seq(-1, 1, length.out = K)
eta <- outer(th, rep(1, K)) - matrix(b0, N, K, byrow = TRUE)
eta[, 2] <- eta[, 2] + 0.8 * z                                  # DIF in item 2
Y <- matrix(rbinom(N * K, 1, plogis(eta)), N)
m <- block_model(Y = Y, z = z,
  theta[person] ~ normal(g * z, 1), g ~ normal(0, 1),
  a[item] ~ normal(1, 1, lower = 0), b[item] ~ normal(0, 3),
  Y ~ bernoulli_logit(a * (theta - b)))
fit <- ecm(m, prior = "ml")
score_test(fit, add = Y ~ delta[item] * z)
```

---

scores *Person scores*

Description

Posterior means (EAP) and posterior SDs of ability and speed, one row per person.

Usage

```
scores(object, ...)

## S3 method for class 'rtirt'
scores(object, ...)
```

Arguments

`object`	A sampled model.
`...`	Unused.

Value

A tibble (class rtirt_tbl: printed with 3 decimals, stored in full precision) with `id`, `ability`, `ability_psd` and (RT models) `speed`, `speed_psd`.

set_cond *Conditions of a simulation (or estimation) run*

Description

Mirrors `setCond()` of ExtendedRtIrtModeling.jl (Julia argument names are accepted by the alias `setCond()`). For real data it is not needed: the dimensions come from the data and the MCMC settings are arguments of `gibbs()` (`gibbs(data, model = ...)` does not use those of `set_cond()`). In simulations it carries the dimensions for `sim_para()` / `sim_data()` and default MCMC settings.

Usage

```
set_cond(
  n_subj = NULL,
  n_item = NULL,
  n_feat = NULL,
  n_iter = 5000,
  n_chain = 4,
  n_burnin = NULL,
  n_thin = 1,
  n_rep = 10,
  q_ra = 0.5,
  q_rt = 0.5
)
```

Arguments

`n_subj`, `n_item`, `n_feat`	Numbers of persons, items and covariates (`NULL`: taken from the data when a model is built).
`n_iter`, `n_chain`	Iterations per chain and number of (independent) chains.
`n_burnin`	Burn-in per chain (default `n_iter / 2`; the Julia version ignored this argument).
`n_thin`	Thinning interval for the stored draws.
`n_rep`	Replications in `run_simulation()`.
`q_ra`	Kept for compatibility (unused, as in the Julia package).
`q_rt`	Quantile level used by the Julia aliases `GibbsRtIrtLatentQr()`, `GibbsRtIrtCrossQr()` (the snake_case constructors take `quantile =`).

Value

An object of class `sim_cond`.

Examples

```
cond <- set_cond(n_subj = 500, n_item = 10, n_feat = 2, n_iter = 2000, n_chain = 3)
cond
```

simulation *True parameters and simulated data*

Description

`sim_para()` draws true parameters for a condition; `sim_data()` simulates data from them (the person parameters are stored in `attr(data, "true_para")`). Item parameters: a ~ N+(1, 0.2^2), b ~ N(0, 0.5^2), sigma2t ~ logN(log 0.3, 0.2^2).

Usage

```
sim_para(cond, model = sim_models, true_corr = 0.3)

sim_data(cond, para, model = sim_models, type = c("norm", "tail", "skew"))
```

Arguments

`cond`	Conditions from `set_cond()`.
`model`	The model of `ecm()` and `gibbs()` to simulate from, by the same name: `"cross"` (default), `"latent"`, `"latreg"`, `"null"` or `"irt"` (the 2PL with a latent regression; `"mlirt"`, its older name, is also accepted). Case does not matter.
`true_corr`	Correlation of ability and speed (`"latreg"`, `"null"`).
`para`	True parameters from `sim_para()` with the same `model`.
`type`	Residual distribution: `"norm"`, `"tail"` (t with 5 df, scaled to the same variance) or `"skew"` (centred Gamma(0.5, 1), scaled to the same variance); `"cross"` (log RT residuals) and `"latent"` (speed residuals).

Value

`sim_para`: an `input_para()`; `sim_data`: an `input_data()` (items `item01`, ...; covariates `x1`, ...) with the true parameters (item and person) in `attr(, "true_para")`, which `compare_para()` uses after `ecm(data, model = ...)` or `gibbs(data, model = ...)`.

summary.rtirt *Summary of a sampled model*

Description

Item parameters (posterior means), the structural parameters with posterior SD, 95% interval, R-hat and effective sample size, the person covariance, reliability and DIC.

Usage

```
## S3 method for class 'rtirt'
summary(object, digits = 3, ...)
```

Arguments

`object`	A sampled model.
`digits`	Decimals to print.
`...`	Unused.

Value

An `rtirt_summary`: a list of the tables (see rtirt_summary), printed.

`vi` *Variational Bayes (deprecated)*

Description

Deprecated: `vi()` still works (with a warning once per session) but gets no new features. Use `ecm()` for estimates or `gibbs()` for the posterior.

Usage

```
vi(model, ...)
```

Arguments

`model`	A `block_model()`.
`...`	Passed to the method.

Details

`vi()` fits a model by mean-field variational Bayes (CAVI); see `vi.block_model()`.

Index

+.normal_part *(*normal_parts*)*, 58

algorithm, 4
algorithm(), *3*, *12*
algorithm.block_model
*(*gibbs.block_model*)*, 45
algorithm.block_model_ecm
*(*ecm.block_model*)*, 31
algorithm.block_model_fit
*(*gibbs.block_model*)*, 45
algorithm.block_model_vi, 6
anova(), *30*
anova.rtirt, 9
anova.rtirt(), *4*
as.character.rtirt_equations
*(*equations*)*, 40
as.data.frame.rtirt *(*estimates*)*, 42
as.data.frame.rtirt_summary
*(*rtirt_summary*)*, 66
as.matrix.block_fit *(*run_sampler*)*, 70

birtRcpp *(*birtRcpp-package*)*, 3
birtRcpp-package, 3
block_model, 10
block_model(), *4*, *6*, *7*, *11*, *16–19*, *21*, *31–33*, *45*, *54*, *55*, *75*, *80*
block_steps, 11, *46*
blocks, *4*, *6*, 12, *23*, *37*
bread.rtirt *(*estfun.rtirt*)*, 41

calc_artificial_data *(*em_components*)*, 37
calc_hessian *(*em_components*)*, 37
calc_item_sums *(*em_components*)*, 37
calc_loglik_nodes *(*em_components*)*, 37
calc_node_weights *(*em_components*)*, 37
calc_nodes *(*em_components*)*, 37
calc_nodes(), *69*
calc_normal_given_nodes
*(*em_components*)*, 37
calc_normal_part *(*compose*)*, 23
calc_normal_part(), *15*, *59*
calc_person_fit *(*compose*)*, 23
check_ecm, 16
check_ecm(), *3*, *21*, *33*
check_sampler, 17
check_sampler(), *3*, *12*, *15*, *16*, *71*
check_vi, 20
checkConvergence *(*julia_aliases*)*, 50
ci_test, 22
ci_test(), *75*
coef(), *67*
coef.rtirt *(*estimates*)*, 42
coef.rtirt_qset *(*rtirt_qset*)*, 65
compare_para, 23
compare_para(), *52*, *67*, *79*
comparePara *(*julia_aliases*)*, 50
compose, 23, *37*
convergence, 26
convergence(), *4*, *20*, *48*, *52*, *65*

dic, 27
dic(), *44*, *52*, *54*, *67*
draw *(*normal_parts*)*, 58
draw(), *15*
draw_cor *(*compose*)*, 23
draw_cor(), *53*
draw_gauss *(*blocks*)*, 12
draw_gauss(), *24*
draw_invgamma *(*blocks*)*, 12
draw_invwishart *(*compose*)*, 23
draw_items_joint *(*blocks*)*, 12
draw_items_joint(), *25*
draw_items_mh *(*compose*)*, 23
draw_items_pg *(*blocks*)*, 12
draw_items_pg(), *69*
draw_lambda *(*blocks*)*, 12
draw_mala *(*mala*)*, 52
draw_mala(), *25*
draw_mean_mvn *(*compose*)*, 23
draw_mvn *(*compose*)*, 23

draw_mvn(), *59*
draw_nu *(*blocks*)*, 12
draw_omega *(*blocks*)*, 12
draw_pg *(*blocks*)*, 12
draw_probit_z *(*blocks*)*, 12
draw_probit_z(), *24*
draw_rt_items_collapsed *(*blocks*)*, 12
draw_shift, 28
draw_sigma2 *(*blocks*)*, 12
draw_theta_collapsed *(*blocks*)*, 12
draw_tnorm *(*blocks*)*, 12
draw_tnorm(), *24*
draw_var *(*blocks*)*, 12

ecm, 29
ecm(), *3–9*, *16–18*, *21, 22*, *26*, *36, 37*, *40–44*, *46, 47*, *56*, *62*, *65, 66*, *73*, *75*, *79, 80*
ecm.block_model, 31
ecm.block_model(), *56*
ecm_control, 36
ecm_control(), *4*, *29, 30*
em_components, 37, *69*
em_squarem *(*em_components*)*, 37
em_squarem(), *69*
equations, 40
equations(), *3, 4*, *6*, *10*
equations.block_model *(*block_model*)*, 10
estep_irt *(*run_em*)*, 68
estfun.rtirt, 41
estfun.rtirt(), *73*
estimates, 42
estimates(), *4*, *26*, *31*, *48*, *67*
estimates.rtirt_qset *(*rtirt_qset*)*, 65

fit_indices, 44
fit_indices(), *4*, *9*, *31*, *37*, *48*, *67*
format.rtirt_tbl *(*rtirt_tbl*)*, 67

get_bias, 44
get_bias(), *52*
get_corr *(*get_bias*)*, 44
get_corr(), *52*
get_rmse *(*get_bias*)*, 44
get_rmse(), *52*
getBias *(*julia_aliases*)*, 50
getCorr *(*julia_aliases*)*, 50
getDic *(*julia_aliases*)*, 50
getRmse *(*julia_aliases*)*, 50
gibbs *(*gibbs.input_data*)*, 46

gibbs(), *3–12*, *14*, *17–19*, *21*, *26*, *30–33*, *36, 37*, *40, 41*, *49*, *52*, *56–58*, *60*, *62, 63*, *71, 72*, *78–80*
gibbs.block_model, 45
gibbs.input_data, 46
gibbs_control *(*ecm_control*)*, 36
gibbs_control(), *4*, *47*
GibbsMlIrt *(*julia_aliases*)*, 50
GibbsRtIrt *(*julia_aliases*)*, 50
GibbsRtIrtCross *(*julia_aliases*)*, 50
GibbsRtIrtCrossQr *(*julia_aliases*)*, 50
GibbsRtIrtLatent *(*julia_aliases*)*, 50
GibbsRtIrtLatentQr *(*julia_aliases*)*, 50
GibbsRtIrtNull *(*julia_aliases*)*, 50
graphics::plot(), *61*, *66*

input_data, 48
input_data(), *4, 5*, *29*, *41*, *47*, *52*, *57*, *79*
input_para, 49
input_para(), *48*, *52*, *58*, *79*
InputData *(*julia_aliases*)*, 50
InputData4R *(*julia_aliases*)*, 50
InputPara *(*julia_aliases*)*, 50
irt_part *(*blocks*)*, 12
irt_part(), *59*

julia_aliases, *4*, 50, *58*

mala, 52
marginal_loglik, 54
marginal_loglik(), *37*, *44*, *67*
mcmc_adapt *(*mala*)*, 52
mcmc_adapt(), *71*
mlirt *(*models*)*, 56
mlirt(), *52*
mml, 54
mml(), *4*, *75*
models, *5*, *41*, 56
mstep_items *(*run_em*)*, 68

normal_part *(*normal_parts*)*, 58
normal_parts, *15*, *25*, 58
normal_prior *(*normal_parts*)*, 58

person_fit, 60
person_fit(), *67*
plot.rtirt, 61
plot.rtirt(), *4*
plot.rtirt_qset *(*rtirt_qset*)*, 65

precis (estimates), 42
precis(), *52*
preknowledge_test, 62
preknowledge_test(), *67*
print.block_em (run_em), 68
print.block_fit (run_sampler), 70
print.block_model (block_model), 10
print.block_model_ecm (ecm.block_model), 31
print.block_model_vi (algorithm.block_model_vi), 6
print.ecm_control (ecm_control), 36
print.em_check (check_ecm), 16
print.gibbs_control (ecm_control), 36
print.normal_part (normal_parts), 58
print.rtirt_algorithm (algorithm), 4
print.rtirt_anova (anova.rtirt), 9
print.rtirt_check (check_sampler), 17
print.rtirt_convergence (convergence), 26
print.rtirt_equations (equations), 40
print.rtirt_precis (estimates), 42
print.rtirt_qset (rtirt_qset), 65
print.rtirt_summary (rtirt_summary), 66
print.rtirt_table (rtirt_summary), 66
print.rtirt_tbl (rtirt_tbl), 67
print.rtirt_values (rtirt_tbl), 67
print.summary.rtirt_qset (rtirt_qset), 65

reg_part (normal_parts), 58
reliability, 63
reliability(), *4*, *31*, *48*, *67*
rtirt_cross (models), 56
rtirt_cross(), *52*
rtirt_latent (models), 56
rtirt_latent(), *52*
rtirt_latreg (models), 56
rtirt_latreg(), *52*
rtirt_null (models), 56
rtirt_null(), *52*
rtirt_priors, 63
rtirt_priors(), *4*, *14*, *15*, *19*, *30*, *47*, *69*
rtirt_priors_julia (rtirt_priors), 63
rtirt_priors_julia(), *64*
rtirt_qset, 65
rtirt_summary, 66, *80*
rtirt_tbl, *43*, *54*, *60*, *62*, 67, *73*, *77*
run_em, 68
run_em(), *7*, *16*, *17*, *32*, *33*, *55*, *71*
run_sampler, 70
run_sampler(), *11*, *15*, *20*, *45*, *46*, *68*
run_simulation, 72
run_simulation(), *52*, *67*, *78*
runSimulation (julia_aliases), 50

sampleGibbs (julia_aliases), 50
score_test, 73
score_test(), *42*
score_test.block_model_ecm, 74
score_test.rtirt(), *76*
scores, 77
scores(), *4*, *31*, *48*, *67*
set_cond, 78
set_cond(), *52*, *57*, *79*
setCond (julia_aliases), 50
setCond(), *78*
setDataMlIrt (julia_aliases), 50
setDataRtIrt (julia_aliases), 50
setDataRtIrtCross (julia_aliases), 50
setDataRtIrtLatent (julia_aliases), 50
setDataRtIrtNull (julia_aliases), 50
setTrueParaMlIrt (julia_aliases), 50
setTrueParaRtIrt (julia_aliases), 50
setTrueParaRtIrtCross (julia_aliases), 50
setTrueParaRtIrtLatent (julia_aliases), 50
sim_data (simulation), 79
sim_data(), *52*, *58*, *78*
sim_para (simulation), 79
sim_para(), *52*, *72*, *78*
simulation, 79
stats::nlminb(), *56*
stats::printCoefmat(), *9*
step_gibbs (block_steps), 11
step_gibbs(), *10*
step_mala (block_steps), 11
step_mala(), *10*
step_shift (block_steps), 11
step_shift(), *10*
summary(), *40*, *52*
summary.block_em (run_em), 68
summary.block_fit (run_sampler), 70
summary.rtirt, 79
summary.rtirt_qset (rtirt_qset), 65

update_items_logistic *(*em_components*)*,
37

vi, 80
vi(), *4*, *20*
vi.block_model
*(*algorithm.block_model_vi*)*, 6
vi.block_model(), *80*