Package 'bggum'

May 7, 2026

Title Bayesian Estimation of Generalized Graded Unfolding Model Parameters

Version 1.0.2

Date 2020-01-18

Description Provides a Metropolis-coupled Markov chain Monte Carlo sampler, post-processing and parameter estimation functions, and plotting utilities for the generalized graded unfolding model of Roberts, Donoghue, and Laughlin (2000) <doi:10.1177/01466216000241001>.

URL https://github.com/duckmayr/bggum

BugReports https://github.com/duckmayr/bggum/issues

Depends R (>= 3.5.0)

Imports stats, graphics, Rcpp (>= 0.12.14)

LinkingTo Rcpp, RcppDist

License GPL (>= 2)

Suggests devtools, testthat, covr, knitr, rmarkdown, dplyr, tidyr

Collate 'bggum-package.R' 'RcppExports.R' 'ggumProbability.R' 'tune_proposals.R' 'tune_temps.R' 'ggumMCMC.R' 'ggumMC3.R' 'ggum_simulation.R' 'color_palettes.R' 'irf.R' 'icc.R' 'summary.R' 'post_process.R'

Encoding UTF-8

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation yes

Author JBrandon Duck-Mayr [aut, cre] (ORCID: <https://orcid.org/0000-0002-2231-1294>), Jacob Montgomery [aut], Patrick Silva [ctb], Luwei Ying [ctb]

Maintainer JBrandon Duck-Mayr <j.duckmayr@gmail.com>

Repository CRAN

Date/Publication 2020-01-19 08:40:02 UTC

Contents

---

bggum *bggum*

Description

bggum provides R tools for the Bayesian estimation of generalized graded unfolding model (Roberts, Donoghue, and Laughlin 2000) parameters. Please see the vignette (via `vignette("bggum")`) for a practical guide to Bayesian estimation of GGUM parameters. Duck-Mayr and Montgomery (2019) provides a more detailed theoretical discussion of Bayesian estimation of GGUM parameters.

Author(s)

JBrandon Duck-Mayr and Jacob Montgomery

References

Duck-Mayr, JBrandon, and Jacob Montgomery. 2019. "Ends Against the Middle: Scaling Votes When Ideological Opposites Behave the Same for Antithetical Reasons." http://jbduckmayr.com/papers/ggum.pdf.

Roberts, James S., John R. Donoghue, and James E. Laughlin. 2000. "A General Item Response Theory Model for Unfolding Unidimensional Polytomous Responses." *Applied Psychological Measurement* 24(1): 3–32.

color_palettes *Color palettes provided by* bggum.

Description

bggum provides color palettes that can be used with its plotting functions. The okabe_ito palette is the eight color, colorblind-friendly palette from Okabe and Ito (2008). The tango palette is comprised of six colors from the Tango palette (Tango Desktop Project 2013).

Usage

```
okabe_ito(n)

tango(n)
```

Arguments

n	An integer vector of length one; the number of colors to return. If n is greater than the number of colors in the palette, the colors will be recycled so that the result is of length n.

Palettes provided

okabe_ito c("#e69f00", "#56b4e9", "#009e73", "#f0e442", "#0072b2", "#d55e00", "#cc79a7", "#000000")

tango c("#cc0000", "#75507b", "#73d216", "#f57900", "#3465a4", "#555753")

References

Okabe, Masataka and Kei Ito. 2008. "Color Universal Design." https://jfly.uni-koeln.de/color/.

Tango Desktop Project. 2013. "Tango Icon Theme Guidelines." https://web.archive.org/web/20160202102503/http://tango.freedesktop.org/Tango_Icon_Theme_Guidelines

Examples

```
## Palettes that are a subset of the total available colors
okabe_ito(3)
tango(3)
## Palettes that need more colors than are available -- leads to recycling
okabe_ito(10)
tango(10)
```

ggumMC3 *GGUM MC3*

Description

Metropolis Coupled Markov Chain Monte Carlo (MC3) Sampling for the GGUM

Usage

```
ggumMC3(data, sample_iterations = 10000, burn_iterations = 10000,
  sd_tune_iterations = 5000, temp_tune_iterations = 5000,
  temp_n_draws = 2500, swap_interval = 1, flip_interval = NA,
  n_temps = length(temps), temps = NULL, optimize_temps = TRUE,
  temp_multiplier = 0.1, proposal_sds = NULL, theta_init = NULL,
  alpha_init = NULL, delta_init = NULL, tau_init = NULL,
  theta_prior_params = c(0, 1), alpha_prior_params = c(1.5, 1.5, 0.25,
  4), delta_prior_params = c(2, 2, -5, 5), tau_prior_params = c(2, 2,
  -6, 6), return_sds = TRUE, return_temps = TRUE)
```

Arguments

`data`	An integer matrix giving the individuals' responses; note the item options should be of the form 0, 1, ... (an example of preparing data for analysis is given in the vignette, available via `vignette("bggum")`)
`sample_iterations`	An integer vector of length one giving the number of iterations the sampler should complete (default is 10000)
`burn_iterations`	An integer vector of length one giving the number of iterations to burn in (default is 10000)
`sd_tune_iterations`	An integer vector of length one; the number of iterations to use to tune the proposals before the burn-in period begins (default is 5000). If 0 is given, the proposals are not tuned.
`temp_tune_iterations`	An integer vector of length one; if a temperature schedule is not provided in the `temps` argument and `optimize_temps` = TRUE, `temp_tune_iterations` gives the number of iterations to use to tune each temperature before the burn-in period begins (default is 5000) – see `tune_temperatures`
`temp_n_draws`	An integer vector of length one; if a temperature schedule is not provided in the `temps` argument and `optimize_temps` = TRUE, `temp_n_draws` gives the number of draws from the temperature finding algorithm to calculate each temperature (default is 2500) – see `tune_temperatures`
`swap_interval`	The period by which to attempt chain swaps; e.g. if swap_interval = 100, a state swap will be proposed between two adjacent chains every 100 iterations (default is 1)

`flip_interval` (Optional) If given, provides the number of iterations after which the sign of the thetas and deltas should be changed. For example, if `flip_interval = 1000`, every 1000 iterations the theta and delta parameters will be multiplied by -1 (a valid parameter value change as discussed in Geyer (1991)).

`n_temps` The number of chains; should only be given if `temps` is not specified

`temps` (Optional) A numeric vector giving the temperatures; if not provided and `optimize_temps = FALSE`, each temperature T_t for t > 1 is given by 1 / (1 + `temp_multiplier` * (t-1)), and T_1 = 1, while if `optimize_temps = TRUE`, the temperature schedule is determined according to an optimal temperature finding algorithm – see `tune_temperatures`

`optimize_temps` A logical vector of length one; if TRUE and a temperature schedule is not provided in the `temps` argument, an algorithm is run to determine the optimal temperature schedule (default is TRUE) – see `tune_temperatures`

`temp_multiplier`
A numeric vector of length one; if a temperature schedule is not provided and `optimize_temps = FALSE`, controls the differences between temperatures as described in the description of the `temps` argument (default is 0.1)

`proposal_sds` (Optional) A list of length four where is element is a numeric vector giving standard deviations for the proposals; the first element should be a numeric vector with a standard deviation for the proposal for each respondent's theta parameter (the latent trait), the second a vector with a standard deviation for each item's alpha (discrimination) parameter, the third a vector with a standard deviation for each item's delta (location) parameter, and the fourth a vector with a standard deviation for each item's tau (option threshold) parameters. If not given, the standard deviations are all set to 1.0 before any tuning begins.

`theta_init` (Optional) Either a numeric vector giving an initial value for each respondent's theta parameter, or a numeric matrix giving an initial value for each respondent's theta parameter for each parallel chain; if not given, the initial values are drawn from the prior distribution

`alpha_init` (Optional) Either a numeric vector giving an initial value for each item's alpha parameter, or a numeric matrix giving an initial value for each item's alpha parameter for each parallel chain; if not given, the initial values are drawn from the prior distribution

`delta_init` (Optional) Either a numeric vector giving an initial value for each item's delta parameter, or a numeric matrix giving an initial value for each item's delta parameter for each parallel chain; if not given, the initial values are drawn from the prior distribution

`tau_init` (Optional) Either a list giving an initial value for each item's tau vector, or a list of lists giving an initial value for each item's tau vector for each parallel chain; if not given, the initial values are drawn from the prior distribution

`theta_prior_params`
A numeric vector of length two; the mean and standard deviation of theta parameters' prior distribution (where the theta parameters have a normal prior; the default is 0 and 1)

`alpha_prior_params`
: A numeric vector of length four; the two shape parameters and a and b values for alpha parameters' prior distribution (where the alpha parameters have a four parameter beta prior; the default is 1.5, 1.5, 0.25, and 4)

`delta_prior_params`
: A numeric vector of length four; the two shape parameters and a and b values for delta parameters' prior distribution (where the delta parameters have a four parameter beta prior; the default is 2, 2, -5, and 5)

`tau_prior_params`
: A numeric vector of length four; the two shape parameters and a and b values for tau parameters' prior distribution (where the tau parameters have a four parameter beta prior; the default is 2, 2, -6, and 6)

`return_sds`
: A logical vector of length one; if TRUE, the proposal standard deviations are stored in an attribute of the returned object named "proposal_sds." The default is TRUE.

`return_temps`
: A logical vector of length one; if TRUE, the temperatures of the parallel chains are stored in an attribute of the returned object named "temps." The default is TRUE.

Details

`ggumMC3` provides `R` implementation of the MC3 algorithm from Duck-Mayr and Montgomery (2019). Some details are provided in this help file, but please see the vignette (via `vignette("bggum")`) for a full in-depth practical guide to Bayesian estimation of GGUM parameters.

Our sampler creates random initial values for the parameters of the model, according to their prior distributions. N parallel chains are run, each at a different inverse "temperature"; the first "cold" chain has an inverse temperature of 1, and each subsequent chain has increasingly lower values (still greater than zero, i.e. fractional values). At each iteration, for each chain, new parameter values are proposed from a normal distribution with a mean of the current parameter value, and the proposal is accepted probabilistically using a Metropolis-Hastings acceptance ratio. The purpose of the chains' "temperatures" is to increase the probability of accepting proposals for chains other than the "cold" chain recorded for inference; the acceptance probability in the Metropolis-Hastings update steps for parameter values are raised to the power of the chain's inverse temperature. After every `swap_intervalth` iteration of the sampler, a proposal is made to swap states between adjacent chains as a Metropolis step. For details, please read the vignette via `vignette("bggum")`, or see Duck-Mayr and Montgomery (2019); see also Gill (2008) and Geyer (1991).

Before burn-in, the standard deviation of the proposal densities can be tuned to ensure that the acceptance rate is neither too high nor too low (we keep the acceptance rate between 0.2 and 0.25). This is done if proposal standard deviations are not provided as an argument and `sd_tune_iterations` is greater than 0.

The temperature schedule can also be tuned using an implementation of the temperature tuning algorithm in Atchadé, Roberts, and Rosenthal (2011). This is done if a temperature schedule is not provided as an argument and `optimize_temps = TRUE`. If a temperature schedule is not provided and `optimize_temps = FALSE`, each temperature T_t for t > 1 is given by 1 / (1 + `temp_multiplier` * (t-1)), and T_1 = 1.

Value

A numeric matrix giving the parameter values at each iteration for the cold chain. The matrix will additionally have classes "ggum" (so that summary.ggum can be called on the result) and "mcmc" with an "mcpar" attribute (so that functions from the coda package can be used, e.g. to assess convergence). If return_sds is TRUE, the result also has an attribute "proposal_sds", which will be a list of length four giving the standard deviations of the proposal densities for the theta, alpha, delta, and tau parameters respectively. If return_temps is TRUE, the result also has an attribute "temps", which will be a numeric vector giving the parallel chains' inverse temperatures.

References

Atchadé, Yves F., Gareth O. Roberts, and Jeffrey S. Rosenthal. 2011. "Towards Optimal Scaling of Metropolis-Coupled Markov Chain Monte Carlo." *Statistics and Computing* 21(4): 555–68.

Duck-Mayr, JBrandon, and Jacob Montgomery. 2019. "Ends Against the Middle: Scaling Votes When Ideological Opposites Behave the Same for Antithetical Reasons." http://jbduckmayr.com/papers/ggum.pdf.

Geyer, Charles J. 1991. "Markov Chain Monte Carlo Maximum Likelihood." In Computing Science and Statistics. Proceedings of the 23rd Symposium on the Interface, edited by E. M. Keramides, 156–63. Fairfax Station, VA: Interface Foundation.

Gill, Jeff. 2008. *Bayesian Methods: A Social and Behavioral Sciences Approach*. 2d ed. Boca Raton, FL: Taylor & Francis.

See Also

ggumProbability, ggumMCMC, tune_temperatures

Examples

```
## NOTE: This is a toy example just to demonstrate the function, which uses
## a small dataset and an unreasonably low number of sampling interations.
## For a longer practical guide on Bayesian estimation of GGUM parameters,
## please see the vignette ( via vignette("bggum") ).
## We'll simulate data to use for this example:
set.seed(123)
sim_data <- ggum_simulation(100, 10, 2)
## Now we can generate posterior draws:
## (for the purposes of example, we use 100 iterations,
## though in practice you would use much more)
draws <- ggumMC3(data = sim_data$response_matrix, n_temps = 2,
                 sd_tune_iterations = 100, temp_tune_iterations = 100,
                 temp_n_draws = 50,
                 burn_iterations = 100, sample_iterations = 100)
```

ggumMCMC *GGUM MCMC Sampler*

Description

MCMC sampler for the generalized graded unfolding model (GGUM), utilizing a Metropolis-Hastings algorithm

Usage

```
ggumMCMC(data, sample_iterations = 50000, burn_iterations = 50000,
  tune_iterations = 5000, flip_interval = NA, proposal_sds = NULL,
  theta_init = NULL, alpha_init = NULL, delta_init = NULL,
  tau_init = NULL, theta_prior_params = c(0, 1),
  alpha_prior_params = c(1.5, 1.5, 0.25, 4), delta_prior_params = c(2,
  2, -5, 5), tau_prior_params = c(2, 2, -6, 6), return_sds = TRUE)
```

Arguments

`data`	An integer matrix giving the response by each respondent to each item; note the item options should be of the form 0, 1, ... (an example of preparing data for analysis is given in the vignette, available via `vignette("bggum")`)
`sample_iterations`	An integer vector of length one; the number of iterations the sampler should store (default is 50000)
`burn_iterations`	An integer vector of length one; the number of "burn-in" iterations to run, during which parameter draws are not stored (default is 50000).
`tune_iterations`	An integer vector of length one; the number of iterations to use to tune the proposals before the burn-in period begins (default is 5000). If 0 is given, the proposals are not tuned.
`flip_interval`	(Optional) If given, provides the number of iterations after which the sign of the thetas and deltas should be changed. For example, if `flip_interval = 1000`, every 1000 iterations the theta and delta parameters will be multiplied by -1 (a valid parameter value change as discussed in Geyer (1991)).
`proposal_sds`	(Optional) A list of length four where is element is a numeric vector giving standard deviations for the proposals; the first element should be a numeric vector with a standard deviation for the proposal for each respondent's theta parameter (the latent trait), the second a vector with a standard deviation for each item's alpha (discrimination) parameter, the third a vector with a standard deviation for each item's delta (location) parameter, and the fourth a vector with a standard deviation for each item's tau (option threshold) parameters. If not given, the standard deviations are all set to 1.0 before any tuning begins.
`theta_init`	(Optional) A numeric vector giving an initial value for each respondent's theta parameter; if not given, the initial values are drawn from the prior distribution

`alpha_init` (Optional) A numeric vector giving an initial value for each item's alpha parameter; if not given, the initial values are drawn from the prior distribution

`delta_init` (Optional) A numeric vector giving an initial value for each item's delta parameter; if not given, the initial values are drawn from the prior distribution

`tau_init` (Optional) A list giving an initial value for each item's tau vector; if not given, the initial values are drawn from the prior distribution

`theta_prior_params`
A numeric vector of length two; the mean and standard deviation of theta parameters' prior distribution (where the theta parameters have a normal prior; the default is 0 and 1)

`alpha_prior_params`
A numeric vector of length four; the two shape parameters and a and b values for alpha parameters' prior distribution (where the alpha parameters have a four parameter beta prior; the default is 1.5, 1.5, 0.25, and 4)

`delta_prior_params`
A numeric vector of length four; the two shape parameters and a and b values for delta parameters' prior distribution (where the delta parameters have a four parameter beta prior; the default is 2, 2, -5, and 5)

`tau_prior_params`
A numeric vector of length four; the two shape parameters and a and b values for tau parameters' prior distribution (where the tau parameters have a four parameter beta prior; the default is 2, 2, -6, and 6)

`return_sds` A logical vector of length one; if TRUE, the proposal standard deviations are stored in an attribute of the returned object named "proposal_sds." The default is TRUE.

Details

`ggumMCMC` provides `R` implementation of an MCMC sampler for the GGUM, based heavily on the algorithm given in de la Torre et al (2006); though the package allows parameter estimation from `R`, the functions are actually written in `C++` to allow for reasonable execution time. Some details are provided in this help file, but please see the vignette (via `vignette("bggum")`) for a full in-depth practical guide to Bayesian estimation of GGUM parameters.

Our sampler creates random initial values for the parameters of the model, according to their prior distributions. At each iteration, new parameter values are proposed from a normal distribution with a mean of the current parameter value, and the proposal is accepted probabilistically using a standard Metropolis-Hastings acceptance ratio. During burn-in, parameter draws are not stored. Before burn-in, the standard deviation of the proposal densities can be tuned to ensure that the acceptance rate is neither too high nor too low (we keep the acceptance rate between 0.2 and 0.25). This is done if proposal standard deviations are not provided as an argument and `sd_tune_iterations` is greater than 0.

Value

A numeric matrix with `sample_iterations` rows and one column for every parameter of the model, so that each element of the matrix gives the value of a parameter for a particular iteration of the MCMC algorithm. The matrix will additionally have classes "ggum" (so that `summary.ggum` can

be called on the result) and "mcmc" with an "mcpar" attribute (so that functions from the coda package can be used, e.g. to assess convergence). If `return_sds` is `TRUE`, the result also has an attribute "proposal_sds", which will be a list of length four giving the standard deviations of the proposal densities for the theta, alpha, delta, and tau parameters respectively.

References

de la Torre, Jimmy, Stephen Stark, and Oleksandr S. Chernyshenko. 2006. "Markov Chain Monte Carlo Estimation of Item Parameters for the Generalized Graded Unfolding Model." *Applied Psychological Measurement* 30(3): 216–232.

Geyer, Charles J. 1991. "Markov Chain Monte Carlo Maximum Likelihood." In Computing Science and Statistics. Proceedings of the 23rd Symposium on the Interface, edited by E. M. Keramides, 156–63. Fairfax Station, VA: Interface Foundation.

Roberts, James S., John R. Donoghue, and James E. Laughlin. 2000. "A General Item Response Theory Model for Unfolding Unidimensional Polytomous Responses." *Applied Psychological Measurement* 24(1): 3–32.

See Also

ggumProbability, ggumMC3

Examples

```
## NOTE: This is a toy example just to demonstrate the function, which uses
## a small dataset and an unreasonably low number of sampling interations.
## For a longer practical guide on Bayesian estimation of GGUM parameters,
## please see the vignette ( via vignette("bggum") ).
## We'll simulate data to use for this example:
set.seed(123)
sim_data <- ggum_simulation(100, 10, 2)
## Now we can generate posterior draws:
## (for the purposes of example, we use 100 iterations,
## though in practice you would use much more)
draws <- ggumMCMC(data = sim_data$response_matrix,
                  tune_iterations = 100,
                  burn_iterations = 100,
                  sample_iterations = 100)
```

---

ggumProbability *GGUM Probability Function*

Description

Calculate the probability of a response according to the GGUM

Usage

```
ggumProbability(response, theta, alpha, delta, tau)
```

Arguments

response	A numeric vector or matrix giving the response(s) for which probability should be calculated.
theta	A numeric vector of latent trait score(s) for respondent(s)
alpha	A numeric vector of discrimination parameter(s)
delta	A numeric vector of location parameter(s)
tau	A numeric vector (if responses to one item are given) or a list (if responses to multiple items are given); the tau parameters for each item is a numeric vector of length K (the number of possible responses) giving the options' threshold parameters; the first element of tau should be zero

Details

The General Graded Unfolding Model (GGUM) is an item response model designed to consider the possibility of disagreement for opposite reasons. This function gives the probability of a respondent's response to a test item given item and respondent parameters. The user can calculate the probability of one particular response to an item, for any number of the possible responses to the item, the probability of a vector of responses (either responses by one person to multiple items, or by multiple people to one item), or the probability of each response in a response matrix.

The probability that respondent i chooses option k for item j is given by

$$\frac{\exp(\alpha_j[k(\theta_i - \delta_j) - \sum_{m=0}^{k} \tau_{jm}]) + \exp(\alpha_j[(2K - k - 1)(\theta_i - \delta_j) - \sum_{m=0}^{k} \tau_{jm}])}{\sum_{l=0}^{K-1}[\exp(\alpha_j[l(\theta_i - \delta_j) - \sum_{m=0}^{l} \tau_{jm}]) + \exp(\alpha_j[(2K - l - 1)(\theta_i - \delta_j) - \sum_{m=0}^{l} \tau_{jm}])]}$$

, where θ_i is i's latent trait parameter, α_j is the item's discrimination parameter, δ_j is the item's location parameter, $\tau_{j0}, \ldots, \tau_{j(K-1)}$ are the options' threshold parameters, and τ_{j0} is 0, K is the number of options for item j, and the options are indexed by $k = 0, \ldots, K - 1$.

Value

A matrix or vector of the same dimensions/length of response.

Note

Please note that items' options should be zero-indexed.

References

de la Torre, Jimmy, Stephen Stark, and Oleksandr S. Chernyshenko. 2006. "Markov Chain Monte Carlo Estimation of Item Parameters for the Generalized Graded Unfolding Model." *Applied Psychological Measurement* 30(3): 216–232.

Roberts, James S., John R. Donoghue, and James E. Laughlin. 2000. "A General Item Response Theory Model for Unfolding Unidimensional Polytomous Responses." *Applied Psychological Measurement* 24(1): 3–32.

Examples

```
## What is the probability of a 1 response to a dichotomous item
## with discrimination parameter 2, location parameter 0, and
## option threshold vector (0, -1) for respondents at -1, 0, and 1
## on the latent scale?
ggumProbability(response = rep(1, 3), theta = c(-1, 0, 1), alpha = 2,
                delta = 0, tau = c(0, -1))
## We can also use this function for getting the probability of all
## observed responses given the data and item and person parameter estimtes.
## Here's an example of that with some simulated data:
## Simulate data with 10 items, each with four options, and 100 respondents
set.seed(123)
sim_data <- ggum_simulation(100, 10, 4)
head(ggumProbability(response = sim_data$response_matrix,
                     theta = sim_data$theta,
                     alpha = sim_data$alpha,
                     delta = sim_data$delta,
                     tau = sim_data$tau))
```

`ggum_simulation` *GGUM Simulation*

Description

Generates randomly drawn item and person parameters, and simulated responses.

Usage

```
ggum_simulation(n, m, K, theta = NULL, alpha = NULL, delta = NULL,
  tau = NULL, theta_params = c(0, 1), alpha_params = c(1.5, 1.5,
  0.25, 4), delta_params = c(2, 2, -5, 5), tau_params = c(1.5, 1.5, -2,
  0))
```

Arguments

`n`	An integer vector of length one giving the number of respondents
`m`	An integer vector of length one giving the number of items
`K`	An integer vector giving the number of options for each item; if the vector is of length one, all m items will have the same number of options.
`theta`	(Optional) A numeric vector of respondents' latent traits; if not given, the values are drawn from a normal distribution whose mean and standard deviation are given by the `theta_params` parameter
`alpha`	(Optional) A numeric vector of items' discrimination parameters; if not given, the values are drawn from a four parameter beta distribution whose parameters are given by `alpha_params`

delta	(Optional) A numeric vector of items' location parameters; if not given, the values are drawn from a four parameter beta distribution whose parameters are given by delta_params
tau	(Optional) A list of numeric vectors giving each item's option thresholds; if not given, the values are drawn from a four parameter beta distribution whose parameters are given by tau_params
theta_params	A numeric vector of length two; the mean and standard deviation of the normal distribution theta is drawn from
alpha_params	A numeric vector of length four; the two shape parameters and a and b values for the four parameter beta distribution alpha is drawn from; the default is 1.5, 1.5, 0.25, and 4
delta_params	A numeric vector of length four; the two shape parameters and a and b values for the four parameter beta distribution delta is drawn from; the default is 2, 2, -5, and 5
tau_params	A numeric vector of length four;the two shape parameters and a and b values for the four parameter beta distribution each tau vector is drawn from; the default is 1.5, 1.5, -2, and 0

Value

A list with five elements; "theta" containing the theta draws, "alpha" containing the alpha draws, "delta" containing the delta draws, "tau" containing the tau draws, and "response_matrix" containing the simulated response matrix.

See Also

ggumProbability

Examples

```
## Simulate data with 10 items, each with four options, and 100 respondents
set.seed(123)
sim_data <- ggum_simulation(100, 10, 4)
str(sim_data)
```

---

icc *Item Characteristic Curve*

Description

Plots item characteristic curves given alpha, delta, and tau parameters.

Usage

```
icc(a, d, t, from = -3, to = 3, by = 0.01, layout_matrix = 1,
  main_title = "Item Characteristic Curve", sub = "",
  color = "black", plot_responses = FALSE, thetas = NULL,
  responses = NULL, response_color = "#0000005f")
```

Arguments

a	A numeric vector of alpha parameters
d	A numeric vector of delta parameters
t	Either a list of numeric vectors for the tau parameters for each option, or a numeric vector if the IRF for only one item is desired – note the first element of each vector should be zero
from	A numeric vector of length one, the lowest theta value to estimate response probabilities for; default is -3
to	A numeric vector of length one, the highest theta value to estimate response probabilities for; default is 3
by	A numeric vector of length one giving the spacing between theta values; default is 0.01
layout_matrix	An integer matrix dictating the layout of the plot; the default is a one-column matrix with one element for each item
main_title	A character vector giving the plots' main titles; default is "Item Characteristic Curve".
sub	An optional character vector of subtitles for the resulting plots, to be pasted onto the main title (helpful for titling individual plots when plotting multiple items' ICCs).
color	The color to plot the ICC line in; default is "black"
plot_responses	A logical vector of length one specifying whether to draw points at the theta estimates of actual responses; default is FALSE
thetas	An optional vector of theta estimates for response drawing; if plot_responses = TRUE and thetas is not provided, an error will be thrown.
responses	An optional matrix or vector (if the ICC for only one item is desired) of responses; if plot_responses = TRUE and responses is not provided, an error will be thrown. NOTE: The lowest response for each item should be 0, not 1.
response_color	The color to plot the response points when plot_responses = TRUE; the default is "#0000005f".

Examples

```
## We'll simulate data to use for these examples:
set.seed(123)
sim_data <- ggum_simulation(100, 10, 4)
## You can plot the ICC for one item:
icc(sim_data$alpha[1], sim_data$delta[1], sim_data$tau[[1]])
## Or multiple items:
icc(sim_data$alpha[1:2], sim_data$delta[1:2], sim_data$tau[1:2], sub = 1:2)
## You can also plot the actual responses over the expected response line:
icc(sim_data$alpha[1], sim_data$delta[1], sim_data$tau[[1]],
    plot_responses = TRUE, responses = sim_data$response_matrix[ , 1],
    thetas = sim_data$theta)
```

irf *Item Response Function*

Description

Plots response functions given alpha, delta, and tau parameters.

Usage

```
irf(a, d, t, from = -3, to = 3, by = 0.01, layout_matrix = 1,
  main_title = "Item Response Function", sub = "",
  option_names = NULL, line_types = NULL, color = "black",
  rug = FALSE, thetas = NULL, responses = NULL, sides = 1,
  rug_colors = "black")
```

Arguments

a	A numeric vector of alpha parameters
d	A numeric vector of delta parameters
t	Either a list of numeric vectors for the tau parameters for each option, or a numeric vector if the IRF for only one item is desired – note the first element of each vector should be zero
from	A numeric vector of length one, the lowest theta value to estimate response probabilities for; default is -3
to	A numeric vector of length one, the highest theta value to estimate response probabilities for; default is 3
by	A numeric vector of length one giving the spacing between theta values; default is 0.01
layout_matrix	An integer matrix dictating the layout of the plot; the default is a one-column matrix with one element for each item
main_title	A character vector giving the plots' main titles; default is "Item Response Function".
sub	An optional character vector of subtitles for the resulting plots, to be pasted onto the main title (helpful for titling individual plots when plotting multiple items' IRFs).
option_names	An optional character vector giving names for the items' options; if NULL, generic names (e.g. "Option 1", "Option 2", etc.) are used
line_types	An optional integer vector specifying lty for each option; if not provided, the first option for each question will have lty = 1, the second will have lty = 2, etc.
color	A specification of the colors to draw lines in. Colors can be specified by either a character vector of colors (either names that R recognizes or hexidecimal specifications) or a function taking a single argument for the number of colors to return. See color_palettes for a list of color palettes provided by bggum. The default is "black" (for all lines).

rug	A logical vector of length one specifying whether to draw a rug of theta estimates; the default is FALSE
thetas	An optional vector of theta estimates for rug drawing; if rug = TRUE and thetas is not provided, an error will be thrown.
responses	An optional matrix or vector (if the IRF for only one item is desired) of responses; if rug = TRUE and responses is not provided, an error will be thrown. NOTE: The lowest response for each item should be 0, not 1.
sides	A vector giving the side(s) to draw the rug on if rug = TRUE; if the vector is of length > 1, the first option for each item will be drawn on the side given by the first element of the vector, the rug for the second option for each item will be drawn on the side given by the second element of the vector, etc.
rug_colors	A vector giving the color(s) to draw the rug in if rug = TRUE or a function taking a single argument for the number of colors to return. See color_palettes for a list of color palettes provided by bggum. The default is "black" (for all rugs).

Examples

```
## We'll simulate data to use for these examples:
set.seed(123)
sim_data <- ggum_simulation(100, 10, 4)
## You can plot the IRF for one item:
irf(sim_data$alpha[1], sim_data$delta[1], sim_data$tau[[1]],
    option_names = 0:3)
## Or multiple items:
irf(sim_data$alpha[1:2], sim_data$delta[1:2], sim_data$tau[1:2],
    option_names = 0:3, sub = 1:2)
## You can plot it in color:
irf(sim_data$alpha[1], sim_data$delta[1], sim_data$tau[[1]],
    option_names = 0:3, color = tango)
## You can also plot a rug of the repsondents' theta estimates with the IRF
irf(sim_data$alpha[1], sim_data$delta[1], sim_data$tau[[1]],
    rug = TRUE, responses = sim_data$response_matrix[ , 1],
    thetas = sim_data$theta, option_names = 0:3)
```

---

post_process *Post-process a Posterior Sample*

Description

Post-process the results of ggumMCMC or ggumMC3 using an artificial identifiability constraint (AIC).

Usage

```
post_process(sample, constraint, expected_sign)
```

Arguments

sample	A numeric matrix of posterior draws as returned by ggumMCMC or ggumMC3.
constraint	An integer vector of length one giving the column number of the parameter to constrain, or a character vector of length one giving the column name for the constraint.
expected_sign	A character vector of length one giving the sign for the constraint; it should be either "-" if the constrained parameter is to be negative or "+" if the constrained parameter is to be positive.

Details

Since under the GGUM the probability of a response is the same for any given choice of theta and delta parameters and the negative of that choice; i.e.

$$Pr(z|\theta, \alpha, \delta, \tau) = Pr(z| - \theta, \alpha, -\delta, \tau),$$

if symmetric priors are used, the posterior has a reflective mode. This function transforms a posterior sample by enforcing a constraint that a particular parameter is of a given sign, essentially transforming it into a sample from only one of the reflective modes if a suitable constraint is chosen; using a sufficiently extreme parameter is suggested.

Please see the vignette (via `vignette("bggum")`) for a full in-depth practical guide to Bayesian estimation of GGUM parameters.

Value

A numeric matrix, the post-processed sample.

See Also

ggumMCMC, ggumMC3

Examples

```
## NOTE: This is a toy example just to demonstrate the function, which uses
## a small dataset and an unreasonably low number of sampling interations.
## For a longer practical guide on Bayesian estimation of GGUM parameters,
## please see the vignette ( via vignette("bggum") ).
## We'll simulate data to use for this example:
set.seed(123)
sim_data <- ggum_simulation(100, 10, 2)
## Now we can generate posterior draws
## (for the purposes of example, we use 100 iterations,
## though in practice you would use much more)
draws <- ggumMC3(data = sim_data$response_matrix, n_temps = 2,
                 sd_tune_iterations = 100, temp_tune_iterations = 100,
                 temp_n_draws = 50,
                 burn_iterations = 100, sample_iterations = 100)
## Then you can post-process the output
processed_draws <- post_process(sample = draws,
```

```
                    constraint = which.min(sim_data$theta),
                    expected_sign = "-")
```

`summary.ggum` *Summarize Posterior Draws for GGUM Parameters*

Description

Summarize the results of ggumMCMC or ggumMC3.

Usage

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

## S3 method for class 'list'
summary(object, ..., combine = TRUE)
```

Arguments

`object`	A numeric matrix of posterior draws as returned by ggumMCMC or ggumMC3, or a list of such matrices.
`...`	Arguments to be passed to or from other methods
`combine`	A logical vector of length one; if `TRUE` and `object` is a list of ggum result objects, the matrices are combined and a summary of the combined sample is given; if `FALSE` and `object` is a list of ggum result objects, each matrix will be summarized individually; and if `object` is not a list, it has no effect. The default is `TRUE`.

Details

This function provides the posterior mean, median, standard deviation, and 0.025 and 0.975 quantiles for GGUM parameters from posterior samples drawn using ggumMCMC or ggumMC3. Please note that the quantiles are calculated using the type 8 algorithm from Hyndman and Fan (1996), as suggested by Hyndman and Fan (1996), rather than the type 7 algorithm that would be the default from R's `quantile()`). Before calling this function, care should be taken to ensure that post-processing has been done if necessary to identify the correct reflective posterior mode, as discussed in the vignette and Duck-Mayr and Montgomery (2019).

Please see the vignette (via `vignette("bggum")`) for a full in-depth practical guide to Bayesian estimation of GGUM parameters.

Value

A list with three elements: estimates (a list of length four; a numeric vector giving the means of the theta draws, a numeric vector giving the means of the alpha draws, a numeric vector giving the means of the delta draws, and a list where the means of the tau draws are collated into a tau estimate vector for each item), sds (a list of length four giving the posterior standard deviations for the theta, alpha, delta, and tau draws), and statistics (a matrix with five columns and one row for each parameter giving the 0.025 quantile, the 0.5 quantile, the mean, the 0.975 quantile, and the standard deviation of the posterior draws for each parameter; please note the quantiles are calculated using the type 8 algorithm from Hyndman and Fan 1996, as suggested by Hyndman and Fan 1996, rather than the type 7 algorithm that would be the default from R's `quantile()`).

If `object` is a list and `combine` is `FALSE`, a list of such lists will be returned.

References

Duck-Mayr, JBrandon, and Jacob Montgomery. 2019. "Ends Against the Middle: Scaling Votes When Ideological Opposites Behave the Same for Antithetical Reasons." http://jbduckmayr.com/papers/ggum.pdf.

Hyndman, R. J. and Fan, Y. 1996. "Sample Quantiles in Packages." *American Statistician* 50, 361–365.

See Also

ggumMCMC, ggumMC3

Examples

```
## NOTE: This is a toy example just to demonstrate the function, which uses
## a small dataset and an unreasonably low number of sampling interations.
## For a longer practical guide on Bayesian estimation of GGUM parameters,
## please see the vignette ( via vignette("bggum") ).
## We'll simulate data to use for this example:
set.seed(123)
sim_data <- ggum_simulation(100, 10, 2)
## Now we can generate posterior draws
## (for the purposes of example, we use 100 iterations,
## though in practice you would use much more)
draws <- ggumMC3(data = sim_data$response_matrix, n_temps = 2,
                 sd_tune_iterations = 100, temp_tune_iterations = 100,
                 temp_n_draws = 50,
                 burn_iterations = 100, sample_iterations = 100)
## Then post-process the output
processed_draws <- post_process(sample = draws,
                                constraint = which.min(sim_data$theta),
                                expected_sign = "-")
## And now we can obtain a summary of the posterior
posterior_summary <- summary(processed_draws)
## It contains all the parameter estimates
str(posterior_summary$estimates)
## As well as the posterior standard deviations
str(posterior_summary$sds)
```

```
## And a matrix of the mean (estimates), median, standard deviations,
## and 0.025 and 0.975 quantiles
head(posterior_summary$statistics)
```

`tune_proposals` *Tune proposal densities*

Description

Tunes the standard deviation for the parameters' proposal densities

Usage

```
tune_proposals(data, tune_iterations, K = NULL, thetas = NULL,
  alphas = NULL, deltas = NULL, taus = NULL,
  theta_prior_params = c(0, 1), alpha_prior_params = c(1.5, 1.5, 0.25,
  4), delta_prior_params = c(2, 2, -5, 5), tau_prior_params = c(2, 2,
  -6, 6))
```

Arguments

`data`	An integer matrix giving the response by each respondent to each item
`tune_iterations`	An integer vector of length one; the number of iterations to complete
`K`	(Optional) A numeric vector with an element for each item giving the number of options for the item; if not provided, it is generated by taking the number of unique options observed in the data
`thetas`	(Optional) A numeric vector giving an initial value for each respondent's theta parameter; if not given, the initial values are drawn from the prior distribution
`alphas`	(Optional) A numeric vector giving an initial value for each item's alpha parameter; if not given, the initial values are drawn from the prior distribution
`deltas`	(Optional) A numeric vector giving an initial value for each item's delta parameter; if not given, the initial values are drawn from the prior distribution
`taus`	(Optional) A list giving an initial value for each item's tau vector; if not given, the initial values are drawn from the prior distribution
`theta_prior_params`	A numeric vector of length two; the mean and standard deviation of theta parameters' prior distribution (where the theta parameters have a normal prior; the default is 0 and 1)
`alpha_prior_params`	A numeric vector of length four; the two shape parameters and a and b values for alpha parameters' prior distribution (where the alpha parameters have a four parameter beta prior; the default is 1.5, 1.5, 0.25, and 4)

`delta_prior_params`
: A numeric vector of length four; the two shape parameters and a and b values for delta parameters' prior distribution (where the delta parameters have a four parameter beta prior; the default is 2, 2, -5, and 5)

`tau_prior_params`
: A numeric vector of length four; the two shape parameters and a and b values for tau parameters' prior distribution (where the tau parameters have a four parameter beta prior; the default is 2, 2, -6, and 6)

Details

This function runs the MCMC algorithm for the number of iterations specified in `tune_iterations`, updating parameter values at each iteration. Every 100 iterations, the function determines how many of the previous 100 iterations resulted in an accepted proposal for each parameter. If the number of acceptances was less than 20, the standard deviation of the proposal for that parameter is decreased by (20 - N) * 0.01, where N is the number of acceptances in the previous 100 iterations. If N is greater than 25, the proposal standard deviation is increased by (N - 25) * 0.01.

Please see the vignette (via `vignette("bggum")`) for a full in-depth practical guide to Bayesian estimation of GGUM parameters.

Value

A list, where each element is a numeric vector; the first element is a numeric vector of standard deviations for the theta parameters' proposals, the second for the alpha parameters, the third for the delta parameters, and the fourth for the tau parameters

Warning

The parameters are updated in place; that is, if you supply objects for the `theta`, `alpha`, `delta`, and `tau` arguments, the objects will not hold the same values after the function is run (in the underlying C++ function, these objects are passed by reference).

Examples

```
## NOTE: This is a toy example just to demonstrate the function, which uses
## a small dataset and an unreasonably low number of tuning interations.
## For a longer practical guide on Bayesian estimation of GGUM parameters,
## please see the vignette ( via vignette("bggum") ).
## We'll simulate data to use for this example
set.seed(123)
sim_data <- ggum_simulation(100, 10, 2)
## Now we can tune the proposal densities
## (for the purposes of example, we use 100 iterations,
## though in practice you would use much more)
proposal_sds <- tune_proposals(data = sim_data$response_matrix,
                               tune_iterations = 100)
```

tune_temperatures *tune_temperatures*

Description

Find Optimal Temperatures for the GGUM MCMCMC Sampler

Usage

```
tune_temperatures(data, n_temps, temp_tune_iterations = 5000,
  n_draws = 2500, K = NULL, proposal_sds = NULL,
  sd_tune_iterations = 5000, theta_prior_params = c(0, 1),
  alpha_prior_params = c(1.5, 1.5, 0.25, 4), delta_prior_params = c(2,
  2, -5, 5), tau_prior_params = c(2, 2, -6, 6))
```

Arguments

data	An integer matrix giving the response by each respondent to each item
n_temps	How many temperatures to make?
temp_tune_iterations	How many iterations should the temperature tuning algorithm run for each temperature? (default is 5000)
n_draws	How many draws should be used to determine each temperature? (specifying n_draws < temp_tune_iterations will result in an error; default is 2500).
K	(Optional) A numeric vector with an element for each item giving the number of options for the item; if not provided, it is generated by taking the number of unique options observed in the data
proposal_sds	(Optional) A list of length four where is element is a numeric vector giving standard deviations for the proposals; the first element should be a numeric vector with a standard deviation for the proposal for each respondent's theta parameter (the latent trait), the second a vector with a standard deviation for each item's alpha (discrimination) parameter, the third a vector with a standard deviation for each item's delta (location) parameter, and the fourth a vector with a standard deviation for each item's tau (option threshold) parameters. If not given, the standard deviations are all set to 1.0 before any tuning begins.
sd_tune_iterations	A numeric vector of length one; if proposal standard deviations are not given, this provides the number of iterations to use to tune the proposals before the temperature finding algorithm begins (default is 5000)
theta_prior_params	A numeric vector of length two; the mean and standard deviation of theta parameters' prior distribution (where the theta parameters have a normal prior; the default is 0 and 1)

`alpha_prior_params`
: A numeric vector of length four; the two shape parameters and a and b values for alpha parameters' prior distribution (where the alpha parameters have a four parameter beta prior; the default is 1.5, 1.5, 0.25, and 4)

`delta_prior_params`
: A numeric vector of length four; the two shape parameters and a and b values for delta parameters' prior distribution (where the delta parameters have a four parameter beta prior; the default is 2, 2, -5, and 5)

`tau_prior_params`
: A numeric vector of length four; the two shape parameters and a and b values for tau parameters' prior distribution (where the tau parameters have a four parameter beta prior; the default is 2, 2, -6, and 6)

Details

Atchadé, Roberts, and Rosenthal (2011) determine the optimal swap-acceptance rate for Metropolis-coupled MCMC and provide an algorithm for building optimal temperature schedules. We implement this algorithm in the context of the GGUM to provide a temperature schedule that should result in approximately 0.234 swap acceptance rate between adjacent chains.

Please see the vignette (via `vignette("bggum")`) for a full in-depth practical guide to Bayesian estimation of GGUM parameters.

Value

A numeric vector of temperatures

References

Atchadé, Yves F., Gareth O. Roberts, and Jeffrey S. Rosenthal. 2011. "Towards Optimal Scaling of Metropolis-Coupled Markov Chain Monte Carlo." *Statistics and Computing* 21(4): 555–68.

See Also

`ggumMCMC`, `ggumMC3`

Examples

```
## NOTE: This is a toy example just to demonstrate the function, which uses
## a small dataset and an unreasonably low number of sampling interations.
## For a longer practical guide on Bayesian estimation of GGUM parameters,
## please see the vignette ( via vignette("bggum") ).
## We'll simulate data to use for this example:
set.seed(123)
sim_data <- ggum_simulation(100, 10, 2)
## Now we can tune the temperature schedule:
## (for the purposes of example, we use 100 iterations,
## though in practice you would use much more)
temps <- tune_temperatures(data = sim_data$response_matrix, n_temps = 5,
                           temp_tune_iterations = 100, n_draws = 50,
                           sd_tune_iterations = 100)
```

Index