

Package: ggdmc (via r-universe)

October 4, 2026

Type Package

Title Hierarchical Bayesian Inference for Choice Response Time Models

Version 0.2.8.9

Date 2025-08-08

Maintainer Yi-Shin Lin <yishinlin001@gmail.com>

Description Provides tools for fitting hierarchical Bayesian models of choice and response time using Differential Evolution Markov Chain Monte Carlo (DE-MCMC) sampling. Designed for cognitive scientists and psychologists, it supports models including the Linear Ballistic Accumulator (LBA) and Diffusion Decision Model (DDM), offering flexible parameter mapping and condition-specific modelling. Implements fast, parallelised C++ routines for large-scale applications in decision modelling. Core functionality includes parameter sampling, simulation, model building, posterior recovery, and convergence diagnostics. Sampling methods follow Turner et al. (2013) <doi:10.1037/a0032222>, Braak (2006) <doi:10.1007/s11222-006-8769-1>, and Hu & Tsui (2010) <doi:10.1016/j.cstda.2008.10.025>. The parameter mapping and condition-specific structure are based on Heathcote et al. (2018) <doi:10.3758/s13428-018-1067-y>.

License GPL (>= 2)

Imports ggdmcModel, ggdmcPrior, ggdmcLikelihood, Rcpp (>= 1.0.7), methods, data.table, matrixStats, lattice

Depends R (>= 3.5.0)

LinkingTo Rcpp (>= 1.0.7), RcppArmadillo (>= 0.10.7.5.0), ggdmcHeaders

Suggests testthat (>= 3.0.0)

RoxygenNote 7.3.2

Encoding UTF-8

NeedsCompilation yes

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

Date/Publication 2025-08-13 03:13:42 UTC

RemoteUrl <https://github.com/yxlin/ggdmc>
RemoteRef HEAD
RemoteSha cee9795cce38dc606ce5bdf5436e7d92b88cdda9

Contents

compare	2
compare_many	4
config-class	5
de_input-class	7
gelman	9
ggdmc	10
initialise_phi	11
initialise_theta	12
plot,posterior-method	14
plot_thetas	15
posterior-class	17
prepare_theta_data	17
prepare_thetas_data	19
RebuildPosterior	20
set_up_new_samples	21
setThetaInput	22
StartSampling	24
summary,posterior-method	29
summary_many	31
theta_input-class	32
theta_input-print	33

Index 34

compare	<i>Compare True Parameters to Estimated Quantiles</i>
---------	---

Description

Compares true parameters to posterior quantiles from a fitted model object and calculates estimation accuracy.

Usage

```
compare(  
  object,  
  start = 1L,  
  end = NULL,  
  ps = NULL,  
  probability = c(0.05, 0.5, 0.975),  
  verbose = TRUE  
)
```

Arguments

object	A fitted instance from the a posterior class
start	First iteration to include in comparison (default: 1)
end	Last iteration to include in comparison (default: NULL uses all samples)
ps	Named vector of true parameter values to compare against estimates
probability	Vector of quantiles to compute (default: c(0.05, 0.5, 0.975))
verbose	Logical Whether to print results (default: TRUE)

Details

This function:

- Extracts quantile summaries from the posterior samples
- Validates parameter names match between true values and estimates
- Computes estimation bias using the median
- Returns (and optionally prints) a comparison matrix

Value

A matrix with rows containing:

- True parameter values
- Estimated quantiles (labeled with percentage)
- Bias (Median - True)

See Also

[compare_many](#) for the function compare multiple posterior objects [summary](#) for the function summarises a posterior object used in the situation of fitting empirical data [summary_many](#) for the function summarises multiple posterior objects

Examples

```
## Not run:
model <- ggdmcModel::BuildModel(
  p_map = list(A = "1", B = "1", mean_v = "M", sd_v = "1", st0 = "1", t0 = "1"),
  match_map = list(M = list(s1 = "r1", s2 = "r2")),
  factors = list(S = c("s1", "s2")),
  constants = c(sd_v = 1, st0 = 0),
  accumulators = c("r1", "r2"),
  type = "lba"
)

fits0 <- ggdmc::StartSampling_subject(sub_dmis[[1]], sub_priors,
  sub_migration_prob = 0.00, thin = 1L, seed = 9032
)
fit0 <- ggdmc::RebuildPosterior(fits0)
options(digits = 2)
```

```
est_phi <- compare(fit0, ps = p_vector)

## End(Not run)
```

compare_many

Compare Multiple Posterior Distributions Against True Values

Description

Computes and compares summary statistics across multiple posterior instances, against user-provided true parameter values.

Usage

```
compare_many(objects, start = 1L, end = NULL, ps = NULL, verbose = TRUE)
```

Arguments

objects	A list of posterior objects from parameter optimisation
start	First iteration to include in comparison (default: 1)
end	Last iteration to include in comparison (default: NULL uses all samples)
ps	Named matrix of true parameter values. Each row represents a participant
verbose	Logical indicating whether to print results (default: TRUE)

Details

This function:

- Computes summary statistics across multiple posterior objects
- Calculates specified quantiles for each parameter in each object
- Optionally compares estimates to true values (when ps is provided)

The function computes:

- Bias (difference between median estimate and true value)
- Comparison tables summarising accuracy

Value

A list containing:

- summary_stats: Array of summary statistics across all objects
- quantiles: Array of requested quantiles across all objects
- bias: Matrix of biases (median - true values) if ps is provided
- comparison: Summary comparison table if ps is provided

Examples

```
## Not run:
options(digits = 2)
est_thetas <- compare_many(fit_thetas, ps = ps)
#           A      B mean_v.false mean_v.true      t0
# Mean 0.106 0.761      0.4326      2.524 0.2755
# True 0.367 0.488      0.2402      2.502 0.3085
# Diff 0.261 -0.274     -0.1924     -0.022 0.0330
# Sd   0.039 0.075      0.1425      0.230 0.0491
# True 0.106 0.093      0.1512      0.167 0.0556
# Diff 0.068 0.018      0.0087     -0.063 0.0065

fits <- fits1
phi <- RebuildHyper(fits)
thetas <- RebuildPosteriors(fits)
result <- compare_many(thetas, ps = ps, verbose = TRUE)
#           a      sz      t0      v      z
# Mean 1.00635 0.284 0.1463 2.6565 0.40525
# True 1.00235 0.251 0.1441 2.6892 0.38429
# Diff -0.00400 -0.033 -0.0022 0.0327 -0.02096
# Sd   0.05936 0.071 0.0155 0.5395 0.00851
# True 0.05884 0.011 0.0186 0.5343 0.00920
# Diff -0.00052 -0.060 0.0031 -0.0052 0.00069

## End(Not run)
```

config-class

MCMC Configuration Class and Constructor

Description

The ‘config’ class collects complete MCMC sampling configurations, including priors, parameter specifications, and Differential Evolution (DE) tuning parameters. The ‘set_configs()’ function provides a convenient constructor for creating configuration objects.

Usage

```
set_configs(
  prior = NULL,
  theta_input = NULL,
  de_input = NULL,
  ncore = 3L,
  seed = NULL
)
```

Arguments

prior	A 'prior' object specifying the model's prior distributions
theta_input	A 'theta_input' object specifying parameter configurations
de_input	A 'de_input' object specifying DE-MCMC tuning parameters
ncore	Integer specifying number of cores to use (default = 3L)
seed	Optional random seed (if NULL, seeds are generated automatically)

Details

The 'config' class integrates all components needed for MCMC sampling:

- **Prior distributions** (from ggdmcPrior)
- **Parameter specifications**
- **DE-MCMC tuning parameters** (jump sizes: rp and gamma_precursor; migration probabilities, etc.)
- **Random number generation** with proper seed management

The constructor 'set_configs()':

- Automatically generates seeds when not specified
- Ensures worker seeds differ from main seed
- Validates core assignments
- Checks consistency between components

Each core runs a true independent 'chain', differing from the 'chains'/'chromosome' used to help the DE-MC sampling to work.

Value

'config' class An S4 object containing complete MCMC configuration

'set_configs()' Returns a validated 'config' object

Slots

prior An object of class 'prior' from ggdmcPrior package containing prior distributions

theta_input An object of class 'theta_input' containing parameter specifications

de_input An object of class 'de_input' containing DE-MCMC tuning parameters

seed An integer vector of random seeds for worker chains

main_seed An integer for the main random seed

core_id An integer vector mapping cores to seeds

Validation Rules

The class includes these validity checks:

- Worker seeds must differ from main seed
- Core IDs must be positive integers
- Components must be properly classed objects

Examples

```
## Not run:
# To create a configuration profile, we first set up a de_input

de_input <- setDEInput(
  sub_migration_prob = 0.00,
  nparameter = as.integer(sub_theta_input@nparameter),
  nchain = as.integer(sub_theta_input@nchain)
)

# Then a theta_input
theta_input <- setThetaInput(nmc = 2L, nchain = 3L, pnames = model@pnames, thin = 1)

# Finally we can use set_configs to create configuration profile
configs <- set_configs(prior = sub_priors, theta_input = theta_input,
  de_input = de_input)

# The default setting is to set up three configurations for three independent chains
# You should ensure that you have three different starting `samples`.
cfg <- configs[[1]]

## End(Not run)
```

de_input-class

*Differential Evolution (DE) MCMC Input Configuration***Description**

Provides configuration objects and methods for Differential Evolution Markov Chain Monte Carlo (DE-MCMC) sampling, including migration probabilities and tuning parameters.

Usage

```
setDEInput(
  pop_migration_prob = 0,
  sub_migration_prob = 0,
  gamma_precursor = 2.38,
  rp = 0.001,
  is_hblocked = FALSE,
  is_pblocked = FALSE,
  nparameter = 5L,
  nchain = 3L,
  pop_debug = FALSE,
  sub_debug = FALSE
)

## S4 method for signature 'de_input'
print(x)
```

Arguments

pop_migration_prob	Population-level migration probability (0-1)
sub_migration_prob	Subject-level migration probability (0-1)
gamma_precursor	Scaling factor for proposal generation (typically 2.38)
rp	Random perturbation size for proposals
is_hblocked	Whether to use block updating for hyperparameters.
is_pblocked	Whether to use block updating for subject parameters.
nparameter	Number of parameters in the model
nchain	Number of DE-MC chains/chromosomes. (Note this is not the true independent chain.)
pop_debug	Logical. If TRUE, enables verbose diagnostic output at the population level for debugging purposes (default = FALSE).
sub_debug	Logical. If TRUE, enables verbose diagnostic output at the subject level for debugging purposes (default = FALSE).
x	A 'de_input' object (for print method)

Details

The 'de_input' class encapsulates key configuration parameters for DE-MCMC sampling:

- Chain migration probabilities control between-chain mixing
- Gamma precursor affects proposal jump sizes
- Random perturbation (rp) adds onto the parameter proposal
- Block updating options improve between chain-mixing for correlated parameters

The migration sampler compares the self likelihood against a (randomly selected) neighbor chain's likelihood. Sometimes, it may trap a chain into a local minimal/maximum. Use it wisely.

Value

'setDEInput()' Returns a 'de_input' class object

'print' method Invisibly returns the input object

Slots

pop_migration_prob	Numeric. Population-level chain migration probability (default = 0.00)
sub_migration_prob	Numeric. Subject-level chain migration probability (default = 0.00)
gamma_precursor	Numeric. Scaling factor for proposal distribution (default = 2.38)
rp	Numeric. Random perturbation factor for proposals (default = 0.001)
is_hblocked	Logical. Whether hyperparameters use block updating (default = FALSE)
is_pblocked	Logical. Whether subject parameters use block updating (default = FALSE)

`nparameter` Integer. Number of parameters in model (default = 5L)
`nchain` Integer. Number of MCMC chains (default = 3L)
`pop_debug` Logical. Whether to print hyper-level debugging information
`sub_debug` Logical. Whether to print subject-level debugging information

Functions

`'setClass("de_input")'` Defines the DE-MCMC configuration class
`'setDEInput()'` Constructor function for creating `'de_input'` objects
`'print'` method Displays a readable summary of DE-MCMC configuration

See Also

[StartSampling](#) for functions using these configurations,

Examples

```

# Create a default configuration
de_config <- setDEInput()

# Custom configuration with higher migration and block updating
de_custom <- setDEInput(
  pop_migration_prob = 0.1,
  sub_migration_prob = 0.05,
  is_hblocked = TRUE,
  nparameter = 7L
)

# Print configuration
print(de_custom)

```

gelman

Brook-Gelman Diagnostic (R -hat)

Description

Calculates potential scale reduction factors for MCMC chains.

Usage

```

gelman(x, start = 1, end = NA, conf = 0.95)

## S4 method for signature 'posterior'
gelman(x, start = 1, end = NA, conf = 0.95)

```

Arguments

x	A posterior object or list of posterior samples
start	First iteration to use
end	Last iteration to use (NA for all)
conf	Confidence level for upper CI (default = 0.95)

Value

A list containing psrf and mpsrf (if requested)

Examples

```
## Not run:
fits <- fits1
phi <- RebuildHyper(fits)
thetas <- RebuildPosteriors(fits)
hat <- gelman(phi)
hat <- gelman(fits[[1]]$phi)
hat <- gelman(fits[[2]]$phi)
hat <- gelman(fits[[3]]$phi)

## End(Not run)
```

ggdmc

Hierarchical Bayesian Inference for Choice Response Time Models

Description

Provides tools for Bayesian inference using differential evolution Markov Chain Monte Carlo (DE-MCMC) sampling for a variety of choice response time models. The package supports flexible parameter mappings, hierarchical model structures, and models including the Linear Ballistic Accumulator (LBA) and Diffusion Decision Model (DDM).

Author(s)

Yi-Shin Lin <yishinlin001@gmail.com>

References

- Lin, Y.-S., & Strickland, L. (2020). Evidence accumulation models with R: A practical guide to hierarchical Bayesian methods. *The Quantitative Methods for Psychology*, 16(2), 133–153.
- Heathcote, A., Lin, Y.-S., Reynolds, A., Strickland, L., Gretton, M., & Matzke, D. (2018). Dynamic models of choice. *Behavior Research Methods*, 50(2), 730–741. <https://doi.org/10.3758/s13428-018-1067-y>

Turner, B. M., & Sederberg, P. B. (2012). Approximate Bayesian computation with differential evolution. *Journal of Mathematical Psychology*, 56(5), 375–385.

ter Braak, C. J. F. (2006). A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: Easy Bayesian computing for real parameter spaces. *Statistics and Computing*, 16(3), 239–249.

Hu, B., & Tsui, K.-W. (2010). Distributed evolutionary Monte Carlo for Bayesian computing. *Computational Statistics & Data Analysis*, 54(3), 688–697. <https://doi.org/10.1016/j.csda.2008.10.025>

initialise_phi	<i>Initialize Hierarchical Parameters (Phi) for MCMC Sampling</i>
----------------	---

Description

Creates initial values for hierarchical parameters (phi) by combining population-level priors with subject-specific parameter estimates.

Usage

```
initialise_phi(
  theta_input,
  priors,
  dmis,
  nmc_idx = 1L,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

theta_input	An S4 object of class ‘ThetaInput’ containing individual-level parameter specifications
priors	List of prior distribution specifications for population-level parameters
dmis	List of data model interface objects (one per subject)
nmc_idx	Iteration index for MCMC chain initialisation (default: 1)
seed	Optional random seed for reproducible initialisation (default: NULL)
verbose	Logical flag for printing initialisation details (default: FALSE)

Details

This initialisation function:

- Draws population-level parameters from hierarchical priors using ‘ggdmcPrior::rprior’
- Generates subject-specific variations around population values

- Ensures parameters respect the hierarchical structure specified in theta_input
- Performs bounds checking on all generated parameters

When ‘verbose=TRUE’, prints:

- Population parameter initialisation values
- Summary statistics of subject-level variations
- Any parameter bounds adjustments made

Value

An S4 object of class ‘Phi’ containing:

- Population-level parameter values drawn from hierarchical priors
- Individual-level parameter variations
- Metadata about the initialisation process

Examples

```
## Not run:
# Basic initialisation
phi <- initialise_phi(theta_input, priors, dmi)

# Reproducible initialisation with seed
phi <- initialise_phi(theta_input, priors, dmi, seed = 123)

# Verbose initialisation showing details
phi <- initialise_phi(theta_input, priors, dmi, verbose = TRUE)

## End(Not run)
```

initialise_theta

Initialise Theta Parameters for MCMC Sampling

Description

Creates initial parameter values (theta) for MCMC sampling by combining prior distributions with likelihood information from the data model instance (DMI).

Usage

```
initialise_theta(
  theta_input,
  priors,
  dmi,
  nmc_idx = 1L,
  seed = NULL,
  verbose = FALSE
)
```

Arguments

theta_input	An S4 object of class 'ThetaInput' containing parameter specifications
priors	a 'prior' object from ggdmcPrior carrying prior distributions.
dmi	Data model instance containing data and experiment design specifications.
nmc_idx	Iteration index for MCMC chain initialisation (default: 1)
seed	Optional random seed for reproducible initialisation (default: NULL)
verbose	Logical flag for printing initialisation details (default: FALSE)

Details

This initialisation function:

- Draws initial values from specified prior distributions using 'ggdmcPrior::rprior'
- Optionally adjusts values based on likelihood computed via ggdmcLikelihood::compute_subject_likelihood
- Handles both fixed and free parameters according to theta_input specifications
- Ensures parameters remain within valid bounds during initialisation

When 'verbose=TRUE', prints internal diagnostic information about:

- Which parameters were initialised from priors
- Any likelihood-based adjustments made parameter bounds checking results

Value

An S4 object of class 'Theta' containing:

- Initial parameter values drawn from priors and adjusted by likelihood
- Metadata about initialisation process
- References to the priors and DMI used

Examples

```
## Not run:
# Basic initialisation
theta <- initialise_theta(theta_input, priors, dmi)

# Reproducible initialisation with seed
theta <- initialise_theta(theta_input, priors, dmi, seed = 123)

# Verbose initialisation showing details
theta <- initialise_theta(theta_input, priors, dmi, verbose = TRUE)

## End(Not run)
```

plot,posterior-method *Plot Posterior Distributions or Traces using Lattice*

Description

Visualise MCMC chains using lattice graphics, showing either trace plots of sampling evolution or density plots of parameter distributions.

Usage

```
## S4 method for signature 'posterior'
plot(
  x,
  y,
  start = 1L,
  end = NULL,
  pll = TRUE,
  den = FALSE,
  subchain = FALSE,
  chains = NA,
  facet_chains = FALSE,
  hide_legend = TRUE,
  ...
)
```

Arguments

x	A posterior object containing MCMC samples
y	Ignored (for consistency with generic plot method)
start	First iteration to include (default: 1)
end	Last iteration to include (default: NULL uses all samples)
pll	Logical indicating whether to plot posterior log-likelihood (TRUE) or parameters (FALSE) (default: TRUE)
den	Logical indicating whether to plot densities (TRUE) or traces (FALSE) when pll=FALSE (default: FALSE)
subchain	Logical indicating whether to subset chains (default: FALSE)
chains	Numeric vector specifying which chains to include when subchain=TRUE (default: NA randomly selects 3 chains)
facet_chains	Logical indicating whether to facet chains vertically when pll=TRUE (default: TRUE)
hide_legend	Logical indicating whether to hide legend (default: TRUE)
...	Additional arguments passed to lattice plotting functions

Details

This method provides three plotting modes using lattice graphics:

1. **Log-likelihood traces** (pll=TRUE): Shows sampling evolution of log-likelihood values with optional chain faceting
2. **Parameter trace plots** (pll=FALSE, den=FALSE): Shows sampling evolution for each parameter with chains colored differently
3. **Parameter densities** (pll=FALSE, den=TRUE): Shows kernel density estimates for each parameter's posterior distribution

The function automatically handles:

- Chain coloring using rainbow palette
- Free y-axis scaling across facets
- Intelligent legend placement
- Subsampling of chains when requested

Value

Invisibly returns the lattice plot object. Primarily called for side effects.

Examples

```
## Not run:
# Plot posterior log-likelihood traces
plot(posterior_obj, pll = TRUE)

# Plot parameter traces (first 500 iterations as burn-in)
plot(posterior_obj, pll = FALSE, start = 501)

# Plot parameter densities for 3 random chains
plot(posterior_obj, pll = FALSE, den = TRUE, subchain = TRUE)

## End(Not run)
```

plot_thetas

Plot Theta Parameters using Lattice Graphics

Description

Visualises theta parameter traces using lattice graphics with faceting by parameter. Provides flexible subject selection options.

Usage

```
plot_thetas(
  x,
  start = 1L,
  end = NULL,
  subchain = FALSE,
  chains = NA,
  hide_legend = TRUE,
  subjects = NULL,
  seed = NULL,
  max_subjects = 50
)
```

Arguments

<code>x</code>	A data frame/table containing columns: Iteration, value, Chain, and s (subject label)
<code>start</code>	First iteration to include (default: 1)
<code>end</code>	Last iteration to include (default: NULL uses all samples)
<code>subchain</code>	Logical Whether to plot a subset of chains (default: FALSE)
<code>chains</code>	Numeric vector specifying which chains to include when subchain=TRUE (default: NA includes all chains)
<code>hide_legend</code>	Logical Whether to hide legend (default: TRUE)
<code>subjects</code>	Number of subjects to plot or vector of specific subjects: - NULL (default): plots all subjects, if less than 5. Plot the first 5, when the subject number exceeds 5. - Integer N: randomly samples N subjects - Character vector: plots specified subjects
<code>seed</code>	(Optional) random seed for reproducible sampling (default: NULL)
<code>max_subjects</code>	(Optional) The maximum number of subjects to plot. If the posterior object contains many subjects, setting this argument to at most max_subjects (default: 50). Useful for avoiding overplotting with large datasets.

Value

Invisibly returns the lattice plot object. Primarily called for side effects.

Examples

```
## Not run:
# Prepare data
fits <- fits1
phi <- RebuildHyper(fits)
thetas <- RebuildPosteriors(fits)

DT <- ggdmc::prepare_thetas_data(fits[[1]]$subject_theta, start = 250)
DT <- ggdmc::prepare_thetas_data(thetas, start = 5000)
p1 <- plot_thetas(DT)
```

```

p1 <- plot_thetas(DT, start = 300, end = 400)
p1 <- plot_thetas(DT, start = 300, end = 400, subjects = 5)
p1 <- plot_thetas(DT, start = 300, end = 400, subjects = as.character(1:10))
p1 <- plot_thetas(DT, start = 300, end = 400, max_subjects = 8)

## End(Not run)

```

posterior-class	<i>An S4 class to represent an object storing posterior samples.</i>
-----------------	--

Description

An S4 class to represent an object storing posterior samples.

Slots

theta posterior samples.
 summed_log_prior summed log prior likelihoods.
 log_likelihoods logged likelihoods
 start the index of starting sample
 npar number of free parameters
 pnames free parameter names
 nmc number of Monte Carlo samples
 thin thinning length
 nchain number of Markov chains facilitating the DE-MCMC sampling

prepare_theta_data	<i>Construct Data Table from the Theta Estiamtes</i>
--------------------	--

Description

Extracts and formats theta parameter samples or log-posterior values from a model object for downstream analysis or plotting.

Usage

```
prepare_theta_data(x, start = 1L, end = NULL, p11 = TRUE)
```

Arguments

<code>x</code>	A model object containing MCMC samples (requires specific slots: <code>@nmc</code> , <code>@nchain</code> , <code>@npar</code> , <code>@summed_log_prior</code> , <code>@log_likelihoods</code> , <code>@theta</code> , <code>@pnames</code>)
<code>start</code>	First iteration to include (default: 1)
<code>end</code>	Last iteration to include (default: NULL uses all samples)
<code>pll</code>	Logical indicating whether to prepare log-posterior values (TRUE) or theta parameters (FALSE) (default: TRUE)

Details

This function prepares MCMC output in a long format suitable for:

- Trace plots and other diagnostics
- Posterior summary statistics
- Model comparison

When `pll=TRUE`, it computes $\text{log-posterior} = \text{log_prior} + \text{log_likelihood}$. When `pll=FALSE`, it extracts all theta parameters.

Value

A data.table with columns:

- Iteration: The MCMC iteration number
- Chain: The chain identifier (factor)
- Parameter: The parameter name (factor)
- value: The sampled values or log-posterior values

Examples

```
## Not run:
fits <- fits1
fit <- RebuildHyper(fits)
fit_thetas <- RebuildPosteriors(fits)
DT <- prepare_theta_data(fit)

## End(Not run)
```

prepare_thetas_data	<i>Prepare Theta Parameters Data from MCMC Samples</i>
---------------------	--

Description

Extracts and formats parameter estimates from a MCMC output, with options for subchain extraction and chain selection.

Usage

```
prepare_thetas_data(x, start = 1L, end = NULL, subchain = FALSE, chains = NA)
```

Arguments

x	A model object containing MCMC samples (requires slots: @theta, @pnames, @nchain, @nmc)
start	First iteration to include (default: 1)
end	Last iteration to include (default: NULL uses all available samples)
subchain	Logical Whether to extract subchain samples (default: FALSE)
chains	Numeric vector specifying which chains to include (default: NA includes all chains)

Details

This function provides flexible extraction of theta parameters:

- Handles both main chains and subchains (when available)
- Allows selection of specific chains
- Supports iteration range selection
- Returns data in tidy format for easy plotting with ggplot2

Value

A data.table in long format with columns:

- Iteration: The MCMC iteration number
- Chain: The chain identifier (factor)
- Parameter: The parameter name (factor)
- value: The sampled parameter values

Examples

```
## Not run:
# Basic usage with main chains
theta_data <- prepare_thetas_data(fit, start = 500)

# Extract subchains instead of main chains
subchain_data <- prepare_thetas_data(fit, subchain = TRUE)

# Specific chains only
chain1_data <- prepare_thetas_data(fit, chains = 1)

# Using with ggplot2
library(ggplot2)
ggplot(
  prepare_thetas_data(fit),
  aes(x = Iteration, y = value, color = Chain)
) +
  geom_line(alpha = 0.7) +
  facet_wrap(~Parameter, scales = "free_y") +
  labs(title = "Theta Parameter Traces")

## End(Not run)
```

RebuildPosterior

Rebuild Posterior Objects from MCMC Chains

Description

These functions reconstruct posterior objects by combining multiple DE-MC chains.

Usage

```
RebuildPosterior(x)
```

```
RebuildHyper(x)
```

```
RebuildPosteriors(x)
```

Arguments

x A list of model fit objects (typically of class ‘posterior’) containing MCMC chains to be combined.

Details

These functions are used to:

- Combine multiple DE-MC chains (e.g., from genetic samplers)

- Reconstruct proper posterior objects after manipulation
- Maintain chain information while aggregating results

The functions handle:

- Combining theta/phi parameters across DE-MC chains
- Aggregating log-prior and log-likelihood matrices
- Properly setting dimensions (nmc, nchain, etc.)

Value

RebuildPosterior Returns a single ‘posterior’ object combining all DE-MC chains from the theta object

RebuildHyper Returns a single ‘posterior’ object from hierarchical model fits

RebuildPosteriors Returns a list of ‘posterior’ objects (one per subject)

Notes

- Input objects should have consistent dimensions across chains
- The ‘thin’ parameter is taken from the first chain
- For hierarchical models, use ‘RebuildPosteriors’ for subject-level parameters and ‘Rebuild-Hyper’ for group-level parameters

Examples

```
## Not run:
# After running multiple MCMC chains:
fits <- fits1
phi <- RebuildHyper(fits)
thetas <- RebuildPosteriors(fits)

## End(Not run)
```

set_up_new_samples	<i>Set Theta Storage for Model</i>
--------------------	------------------------------------

Description

Creates and configures a theta container for use in model fitting.

Usage

```
set_up_new_samples(theta_input)
```

Arguments

theta_input An S4 object containing parameters specifying the dimension and other information for the storage.

Details

This function takes the configuration parameter needed to create a posterior

Value

An S4 object of class ‘posterior’ containing:

- The configured theta parameters
- Additional metadata needed for posterior calculations

Examples

```
## Not run:
result <- set_up_new_samples(theta_input)

## End(Not run)
```

setThetaInput	<i>Create a theta_input Object for MCMC Configuration</i>
---------------	---

Description

Constructs a theta_input instance storing the parameter that controls Markov Chain Monte Carlo (MCMC) sampling.

Usage

```
setThetaInput(
  nmc = 500L,
  nchain = NULL,
  thin = 1L,
  pnames = NULL,
  report_length = 100L,
  max_init_attempts = 1000L,
  is_print = TRUE
)
```

Arguments

nmc	Integer specifying total number of MCMC iterations (including burn-in). Default: 500L
nchain	Integer specifying number of MCMC chains facilitating the DE-MC sampling. If NULL (default), it will be set to 3 times the number of parameters.
thin	Integer specifying thinning interval (keep every nth sample). Default: 1L (no thinning)
pnames	Character vector naming the free parameters. Default: NULL
report_length	Integer controlling how often progress is reported. Default: 100L (report every 100 iterations)
max_init_attempts	Integer specifying maximum attempts to initialise valid starting values. Default: 1000L
is_print	Logical controlling whether to print sampling progress. Default: TRUE

Details

This constructor:

- Validates all input parameters
- Automatically sets nchain to $3 \times$ number of parameters if NULL
- Derives nparameter from length of pnames
- Ensures all numeric parameters are converted to integers

Value

A theta_input S4 object containing MCMC configuration parameters with the following slots:

- nmc: Total iterations
- nchain: Number of chains
- thin: Thinning interval
- nparameter: Number of parameters (derived from pnames)
- pnames: Parameter names
- report_length: Progress report frequency
- max_init_attempts: Initialisation attempts
- is_print: Print progress flag

Examples

```
# A minimal LBA model with 5 free parameters
nmc <- 1000L
npar <- 5L
nchain <- npar * 3
thin <- 1L
pnames <- c("A", "B", "mean_v.false", "mean_v.true", "t0")
```

```

sub_theta_input <- setThetaInput(
  nmc = nmc, nchain = as.integer(nchain), pnames = pnames,
  thin = thin
)

# theta_input has its own print method
print(sub_theta_input)

```

StartSampling

Initialise or Continue MCMC Sampling

Description

These functions handle initialisation and continuation of Markov Chain Monte Carlo (MCMC) sampling for hierarchical models, individual subjects, and (merely) hyperparameter estimation (i.e., standard models).

Usage

```

StartSampling(
  dms,
  priors,
  samples_list = NULL,
  nmc = 500L,
  nchain = NULL,
  thin = 1L,
  report_length = 100L,
  max_init_attempts = 1000L,
  is_print = TRUE,
  pop_migration_prob = 0,
  sub_migration_prob = 0,
  gamma_precursor = 2.38,
  rp = 0.001,
  is_hblocked = FALSE,
  is_pblocked = FALSE,
  pop_debug = FALSE,
  sub_debug = FALSE,
  ncore = 3L,
  seed = NULL
)

```

```

RestartSampling(
  samples_list,
  nmc = 500L,
  nchain = NULL,
  thin = 1L,

```

```
    report_length = 100L,  
    max_init_attempts = 1000L,  
    is_print = TRUE,  
    pop_migration_prob = 0,  
    sub_migration_prob = 0,  
    gamma_precursor = 2.38,  
    rp = 0.001,  
    is_hblocked = FALSE,  
    is_pblocked = FALSE,  
    pop_debug = FALSE,  
    sub_debug = FALSE,  
    seed = NULL  
)
```

```
StartSampling_subject(  
  dmi,  
  priors,  
  samples_list = NULL,  
  nmc = 500L,  
  nchain = NULL,  
  thin = 1L,  
  report_length = 100L,  
  max_init_attempts = 1000L,  
  is_print = TRUE,  
  sub_migration_prob = 0,  
  gamma_precursor = 2.38,  
  rp = 0.001,  
  is_pblocked = FALSE,  
  sub_debug = FALSE,  
  ncore = 3L,  
  seed = NULL  
)
```

```
RestartSampling_subject(  
  samples_list,  
  nmc = 500L,  
  thin = 1L,  
  report_length = 100L,  
  max_init_attempts = 1000L,  
  is_print = TRUE,  
  sub_migration_prob = 0,  
  gamma_precursor = 2.38,  
  rp = 0.001,  
  is_pblocked = FALSE,  
  sub_debug = FALSE,  
  seed = NULL  
)
```

```

StartSampling_hyper(
  hyper_dmi,
  dmi,
  priors,
  samples_list = NULL,
  nmc = 500L,
  nchain = NULL,
  thin = 1L,
  report_length = 100L,
  max_init_attempts = 1000L,
  is_print = TRUE,
  sub_migration_prob = 0,
  gamma_precursor = 2.38,
  rp = 0.001,
  is_pblocked = FALSE,
  sub_debug = FALSE,
  ncore = 3L,
  seed = NULL
)

```

```

RestartSampling_hyper(
  samples_list,
  nmc = 500L,
  nchain = NULL,
  thin = 1L,
  report_length = 100L,
  max_init_attempts = 1000L,
  is_print = TRUE,
  sub_migration_prob = 0,
  gamma_precursor = 2.38,
  rp = 0.001,
  is_pblocked = FALSE,
  sub_debug = FALSE,
  seed = NULL
)

```

Arguments

<code>dmi</code>	For hierarchical models: A list of data model instances (DMIs) for each subject
<code>priors</code>	A list of prior distributions
<code>samples_list</code>	Either: • For Start* functions: Optional the user-initilised samples. • For Restart* functions: Required previous samples object to continue sampling
<code>nmc</code>	Number of MCMC iterations to run (default = 500)
<code>nchain</code>	Number of independent chains (default = NULL for automatic determination; not used in RestartSampling_subject)

<code>thin</code>	Thinning interval (default = 1)
<code>report_length</code>	Frequency of progress reports in iterations (default = 100)
<code>max_init_attempts</code>	Maximum attempts to find valid initial values (default = 1000)
<code>is_print</code>	Whether to print progress messages (default = TRUE)
<code>pop_migration_prob</code>	Population-level chain migration probability (default = 0)
<code>sub_migration_prob</code>	Subject-level chain migration probability (default = 0)
<code>gamma_precursor</code>	Scaling factor for proposal distribution (default = 2.38)
<code>rp</code>	Random perturbation factor to add noise to DE-MC proposal. The large the value, the big the proposal will jump (default = 0.001)
<code>is_hblocked</code>	Whether to use block updating for hyperparameters (default = FALSE)
<code>is_pblocked</code>	Whether to use block updating for participant/subject parameters (default = FALSE)
<code>pop_debug</code>	Logical. Whether to print population level debugging information.
<code>sub_debug</code>	Logical. Whether to print subject level debugging information.
<code>ncore</code>	Number of CPU cores for parallel computation (default = 3)
<code>seed</code>	Random seed for reproducibility (default = NULL for random seed)
<code>dmi</code>	For single-subject models: A DMI instance for one subject
<code>hyper_dmi</code>	For standard models: a DMI for hyperparameters using 'type', 'hyper')

Details

Function Types:

- **Hierarchical Models:**
 - 'StartSampling': initialise full hierarchical sampling
 - 'RestartSampling': Continue hierarchical sampling
- **Single-Subject Models:**
 - 'StartSampling_subject': initialise single-subject sampling
 - 'RestartSampling_subject': Continue single-subject sampling
- **Hyperparameter Models:**
 - 'StartSampling_hyper': initialise hyperparameter-only sampling
 - 'RestartSampling_hyper': Continue hyperparameter-only sampling

Key Features:

- Adaptive MCMC with optional chain migration
- Parallel chain execution
- Progress reporting and convergence monitoring
- Three-level modeling (hyperparameter-only, population, and individual subjects)

Value

An object containing MCMC samples, with class:

- ‘hierarchical_samples’ for hierarchical versions
- ‘subject_samples’ for single-subject versions
- ‘hyper_samples’ for hyperparameter versions

Migration Parameters

The migration sampler helps to improve chain mixing at different levels:

- ‘pop_migration_prob’: For population-level parameters
- ‘sub_migration_prob’: For subject-level parameters and hyper-only parameters.

. The sampler essentially compares the self chain against its neighbouring chains, and updates the self chain when the neighboring chain results in higher likelihood. The user must be aware that such a strategy may result in an optimisation process falls into a local maximum (likelihood).

See Also

[RebuildPosterior](#) for combining chains

Examples

```
## Not run:
# Hierarchical models
hier_fits0 <- StartSampling(pop_dmis, pop_priors,
  sub_migration_prob = 0.05,
  thin = 8L, pop_debug = F, seed = 9032
)

hier_fits1 <- RestartSampling(hier_fits0,
  pop_migration_prob = 0.02,
  sub_migration_prob = 0.00,
  thin = 4L, seed = 9032
)

# Single-subject models
subj_fits0 <- StartSampling_subject(sub_dmis[[1]], sub_priors,
  sub_migration_prob = 0.02,
  thin = 2, seed = 9032
)

subj_fits1 <- RestartSampling_subject(subj_fits0,
  sub_migration_prob = 0.00, thin = 4, seed = 9032
)

# Hyperparameter models
hyper_fits0 <- StartSampling_hyper(hyper_dmi, pop_dmis, pop_priors,
  sub_migration_prob = 0.05, thin = 4
)
```

```
hyper_fits1 <- RestartSampling_hyper(fits0, sub_migration_prob = 0.00, thin = 2)

## End(Not run)
```

summary.posterior-method

Summarise Posterior Distribution

Description

Computes summary statistics for samples from a posterior distribution object.

Usage

```
## S4 method for signature 'posterior'
summary(object, start = 1L, end = NULL, probability = c(0.05, 0.5, 0.975))
```

Arguments

object	A posterior class object containing MCMC samples
start	First iteration to include in summary (default: 1)
end	Last iteration to include in summary (default: NULL uses all samples)
probability	Vector of probabilities for quantiles (default: c(0.05, 0.5, 0.975))

Details

This method provides a statistical summary of the ‘theta’ in the posterior ‘samples’:

- Calculates standard statistics (mean, median, SD) for all parameters
- Computes user-specified quantiles
- Handles burn-in through the start and end parameters

The default quantiles (2.5

Value

A list containing:

- statistics: Matrix of summary statistics (mean, sd, etc.)
- quantiles: Matrix of requested quantiles
- start: First iteration used
- end: Last iteration used

Examples

```
## Not run:

# Assuming you have set up DMI, priors etc.
# First make sure that the subject level has converged
fits0 <- StartSampling(pop_dmis, pop_priors,
  sub_migration_prob = 0.05,
  thin = 8L, pop_debug = F, seed = 9032
)

# Then turn on the migration sampler at the population level
fits1 <- RestartSampling(fits0,
  pop_migration_prob = 0.02,
  sub_migration_prob = 0.00,
  thin = 4L, seed = 9032
)

# Turn down the migration sampler, so that we may not fall
# into local minimal
fits2 <- RestartSampling(fits1,
  pop_migration_prob = 0.01,
  sub_migration_prob = 0.00,
  thin = 2L, seed = 9032
)

fits <- fits2
phi <- RebuildHyper(fits)
thetas <- RebuildPosteriors(fits)
est_phi <- summary(phi)

# str(est_phi)
# List of 2
# $ statistics: num [1:10, 1:4] 1.024 0.246 0.15 2.283 0.406 ...
# .. attr(*, "dimnames")=List of 2
# .. ..$ : chr [1:10] "loc_a" "loc_sz" "loc_t0" "loc_v" ...
# .. ..$ : chr [1:4] "Mean" "SD" "Naive SE" "Time-series SE"
# $ quantiles : num [1:10, 1:3] 0.9614 0.0266 0.134 0.2257 0.2275 ...
# .. attr(*, "dimnames")=List of 2
# .. ..$ : chr [1:10] "loc_a" "loc_sz" "loc_t0" "loc_v" ...
# .. ..$ : chr [1:3] "5%" "50%" "97.5%"
# - attr(*, "start")= int 1
# - attr(*, "end")= int 15000
# - attr(*, "thin")= int 4
# - attr(*, "nchain")= int 3

post_summary <- summary(phi, start = 501) # Discard first 500 as burn-in

# Custom quantiles
detailed_summary <- summary(phi, probability = seq(0.1, 0.9, by = 0.1))

Extract specific elements
posterior_means <- post_summary$statistics[, "Mean"]
```

```
credible_intervals <- post_summary$quantiles[, c("5%", "97.5%")]

## End(Not run)
```

summary_many

Summarise Multiple Posterior Objects

Description

Summarise Multiple Posterior Objects

Usage

```
summary_many(objects, start = 1L, end = NULL, verbose = FALSE)
```

Arguments

objects	A list of posterior objects to summarise
start	First iteration to include in summary
end	Last iteration to include in summary (NULL for all)
verbose	Logical Whether to print summarised results. Default: FALSE

Value

Either a list of summary objects (if verbose=FALSE) or a matrix of means

Examples

```
## Not run:
# Assuming you have set up DMI, priors etc.
# First make sure that the subject level has converged
fits0 <- StartSampling(pop_dmis, pop_priors,
  sub_migration_prob = 0.05,
  thin = 8L, pop_debug = F, seed = 9032
)

# Then turn on the migration sampler at the population level
fits1 <- RestartSampling(fits0,
  pop_migration_prob = 0.02,
  sub_migration_prob = 0.00,
  thin = 4L, seed = 9032
)

# Turn down the migration sampler, so that we may not fall
# into local minimal
fits2 <- RestartSampling(fits1,
  pop_migration_prob = 0.01,
  sub_migration_prob = 0.00,
```

```

      thin = 2L, seed = 9032
    )

    fits <- fits2
    phi <- RebuildHyper(fits)
    thetas <- RebuildPosteriors(fits)
    result <- summary_many(thetas)

    # Return all details for every participant
    result <- summary_many(thetas, verbose = TRUE)

    ## End(Not run)

```

theta_input-class	<i>An S4 class to store MCMC sampling parameters.</i>
-------------------	---

Description

This class holds parameters that control the Markov Chain Monte Carlo (MCMC) sampling process.

Details

This class is used to encapsulate the key parameters that govern the MCMC sampling process. These parameters are essential for controlling the length of the MCMC run, how often to keep samples, and the storage and reporting of the sampling process.

Slots

nmc A integer specifying the total number of MCMC iterations (including burn-in).

nchain A integer indicating the number of MCMC chains to run.

thin A integer indicating the thinning interval. This determines how often samples are kept (e.g., a ‘thin’ of 10 keeps every 10th sample). Thinning can help reduce autocorrelation in the MCMC samples.

nparameter A integer storing the number of free parameter.

pnames A string vector storing the name of the free parameter.

report_length A integer specifying the interval at which progress reports are printed during the MCMC run. When running parallel cores on Windows, the print function may not work.

max_init_attempts A integer specifying the maximum number of attempts to initialise an MC sampler via the `rprior()` random number generation.

is_print A logical value specifying whether to print out the progress. upon previous samples.

Objects from the Class

An `theta_input` instance can be created using the ‘`setThetaInput`’ or the conventional ‘`new("theta_input", ...)`’ constructor, where ‘...’ are named arguments corresponding to the slots.

theta_input-print	<i>Print Method for theta_input Objects</i>
-------------------	---

Description

Displays a human-readable summary of a theta_input object's contents.

Usage

```
## S4 method for signature 'theta_input'
print(x)
```

Arguments

x	A theta_input object to be printed
---	------------------------------------

Details

This method provides a concise console representation of theta_input objects, showing all key configuration parameters in a standardised format:

- Basic MCMC dimensions (iterations, chains, thinning)
- Parameter information (count and names)
- Runtime control settings (reporting, initialization attempts)
- Operational flags (printing, accumulation)

The output is designed to give users immediate visibility into the MCMC configuration while avoiding overwhelming technical details.

Value

Invisibly returns the input object x. Called for its side effect of printing to the console.

Examples

```
# Create and print a theta_input object
theta <- setThetaInput(
  nmc = 1000L,
  nchain = 3L,
  pnames = c("alpha", "beta", "sigma")
)
print(theta)

# typing the object name to use the default print method
theta
```

Index

- * **package**
 - ggdmc, 10
- compare, 2
- compare_many, 3, 4
- config-class, 5
- de_input-class, 7
- gelman, 9
 - gelman,posterior-method (gelman), 9
 - gelman-methods (gelman), 9
 - ggdmc, 10
 - ggdmc-package (ggdmc), 10
- initialise_phi, 11
- initialise_theta, 12
- plot,posterior-method, 14
- plot_thetas, 15
- posterior-class, 17
- prepare_theta_data, 17
- prepare_thetas_data, 19
- print,de_input-method (de_input-class), 7
- print,theta_input-method (theta_input-print), 33
- RebuildHyper (RebuildPosterior), 20
- RebuildPosterior, 20, 28
- RebuildPosteriors (RebuildPosterior), 20
- RestartSampling (StartSampling), 24
- RestartSampling_hyper (StartSampling), 24
- RestartSampling_subject (StartSampling), 24
- set_configs (config-class), 5
- set_up_new_samples, 21
- setDEInput (de_input-class), 7
- setThetaInput, 22
- StartSampling, 9, 24
- StartSampling_hyper (StartSampling), 24
- StartSampling_subject (StartSampling), 24
- summary, 3
- summary,posterior-method, 29
- summary_many, 3, 31
- theta_input-class, 32
- theta_input-print, 33