Package: bayesintervals (via r-universe)

November 26, 2026

Title Reducible Imprecision and Bayesian Intervals for Mixed Models

Version 3.0.0

Description Calculates and visualizes Bayesian credible, prediction and tolerance intervals, and posterior predictive distributions, for linear mixed models from Markov chain Monte Carlo (MCMC) samples of the model parameters. Users define the model through fixed effects, variance components and optional covariates, and results can be back-transformed from the log scale. Tolerance intervals are derived from the posterior predictive distribution following Krishnamoorthy and Mathew (2009) <doi:10.1002/9780470473900>, including the construction of one-dimensional two-sided intervals from the bivariate posterior of the lower and upper quantiles. Reducible imprecision measures the share of the tolerance-interval half-width that exceeds the prediction-interval half-width, that is, the epistemic uncertainty that more data could reduce. The package includes an interactive 'shiny' application, started with start_app_view(), which has its own name: 'BSI' (Bayesian Statistical Intervals). The development repository on 'GitHub' is called 'BayesianSI.Calculator'.

License Apache License (== 2.0)

URL https://sanofi-public.github.io/BayesianSI.Calculator/, https://github.com/Sanofi-Public/BayesianSI.Calculator

BugReports https://github.com/Sanofi-Public/BayesianSI.Calculator/issues

Encoding UTF-8

Language en-US

Imports shiny, DT, bslib, later, dplyr, readr, data.table, ggplot2, plotly, stats, utils, grDevices

Depends R (>= 4.1.0)

Suggests testthat

LazyData true

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Ekin Ören [aut, cre], Sebastian Schmitz [aut], Sanofi [cph, fnd]

Maintainer Ekin Ören <ekin.oren.3@gmail.com>

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

Date/Publication 2026-11-26 01:51:39 UTC

RemoteUrl https://github.com/cran/bayesintervals

RemoteRef HEAD

RemoteSha dd8e51b46c77f31837991cb247efc699196d8d3f

Contents

add_krishnamoorthy_ti . . . 3
bayesian_statistical_intervals . . . 4
calculate_ci . . . 6
calculate_gamma_difference . . . 7
calculate_gP_mean . . . 8
calculate_PI . . . 9
calculate_predicted_distributions . . . 10
calculate_tolerance_interval . . . 11
concatenate_by_columns . . . 12
create_all_cdf_dataframes . . . 13
create_CDF . . . 14
create_cdf_dataframe . . . 16
data_InputFileTester2 . . . 17
find_optimal_point . . . 18
format_intervals_table . . . 19
get_probability_table . . . 20
load_n_inspect_data . . . 21
lognorm_adjuster . . . 22
parse_x_value_input . . . 23
perform_calculations . . . 24
plot_intervals . . . 25
plot_krishnamoorthy . . . 26
plot_one_sided_intervals . . . 27
plot_posterior_predictive . . . 29
quantile_scatterplot . . . 30
start_app_view . . . 32

Index **33**

`add_krishnamoorthy_ti` *Add Krishnamoorthy Tolerance Interval Values*

Description

This function adds Krishnamoorthy TI values as new columns (K_TI_Lower and K_TI_Upper) to the data frame, preserving the original TI values.

Usage

```
add_krishnamoorthy_ti(data, optimal_results)
```

Arguments

`data`	A data frame containing at least the following columns: • By: A grouping variable • TI_Lower: Original lower tolerance interval bound • TI_Upper: Original upper tolerance interval bound
`optimal_results`	A list of optimal results from find_optimal_point()

Value

A data frame with additional columns:

- K_TI_Lower: Krishnamoorthy lower tolerance interval bound
- K_TI_Upper: Krishnamoorthy upper tolerance interval bound

References

Krishnamoorthy K, Mathew T (2009). *Statistical Tolerance Regions: Theory, Applications, and Computation*. Wiley, Hoboken, NJ. doi:10.1002/9780470473900.

Examples

```
data <- data.frame(By = c("A", "B"), TI_Lower = c(-3, -4), TI_Upper = c(3, 4))
optimal_results <- list(
  A = list(By = "A", g1 = -2.5, g2 = 2.5),
  B = list(By = "B", g1 = -3.5, g2 = 3.5)
)
add_krishnamoorthy_ti(data, optimal_results)
```

`bayesian_statistical_intervals` *Perform Bayesian Statistical Interval Calculations*

Description

This function coordinates the calculation of various statistical intervals including predicted distributions, credible intervals, tolerance intervals, and prediction intervals.

Usage

```
bayesian_statistical_intervals(
  data,
  fixed_effects,
  random_params,
  by = "Single Chain Dataset",
  tolerance_interval_level,
  multiplication_factor = 1,
  credible_interval_level = 95,
  prediction_interval_level = 90,
  credibility_of_tolerance_interval = 95,
  log_normal = FALSE,
  covariate_cols = NULL,
  covariate_values = NULL,
  use_krishnamoorthy = FALSE,
  krishnamoorthy_tolerance = 0.01
)
```

Arguments

`data`	A data frame containing the input data
`fixed_effects`	Character vector of column names for fixed effects
`random_params`	Character vector of column names for random parameters
`by`	Character, name of the column to group by (default = "Single Chain Dataset")
`tolerance_interval_level`	Numeric, tolerance interval level percentage (between 0 and 100)
`multiplication_factor`	Numeric, factor to multiply random parameters variance by (default = 1)
`credible_interval_level`	Numeric, percentage for credible intervals (default = 95)
`prediction_interval_level`	Numeric, percentage for prediction intervals (default = 90)
`credibility_of_tolerance_interval`	Numeric, credibility level for tolerance intervals (default = 95)
`log_normal`	Logical, whether to transform results from log space (default = FALSE)

`covariate_cols` Character vector of column names for covariates (default = NULL)

`covariate_values`
Numeric vector of values to multiply with covariates (default = NULL)

`use_krishnamoorthy`
Logical, whether to calculate two-sided tolerance intervals using the Krishnamoorthy method (default = FALSE)

`krishnamoorthy_tolerance`
Numeric, convergence tolerance for Krishnamoorthy method (default = 0.01)

Details

When `use_krishnamoorthy=TRUE`, the function adds two additional columns to the output data:

- K_TI_Lower: Lower tolerance interval bound calculated using Krishnamoorthy method
- K_TI_Upper: Upper tolerance interval bound calculated using Krishnamoorthy method

The Krishnamoorthy method provides an alternative approach to calculating two-sided tolerance intervals that may be more appropriate in some situations, particularly when the distribution of lower and upper quantiles is not symmetric. The method finds optimal points (g1, g2) that satisfy the constraint g2 = 2*gP_mean - g1, where gP_mean is the average midpoint of the lower and upper quantiles.

Value

A list containing:

- data: A data frame with calculated intervals and reducible imprecision metrics
- summary: A string summarizing the calculation parameters
- krishnamoorthy_results: If use_krishnamoorthy=TRUE, a list of Krishnamoorthy method results

References

Wolfinger RD (1998). Tolerance intervals for variance component models using Bayesian simulation. *Journal of Quality Technology*, 30(1).

Krishnamoorthy K, Mathew T (2009). *Statistical Tolerance Regions: Theory, Applications, and Computation*. Wiley, Hoboken, NJ. doi:10.1002/9780470473900.

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
)

res <- bayesian_statistical_intervals(
```

```
  mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_interval_level = 90
)
res$data
res$summary

# With a covariate and the Krishnamoorthy two-sided tolerance interval
res_k <- bayesian_statistical_intervals(
  mcmc, fixed_effects = "beta", random_params = "sigma2", by = "Group",
  tolerance_interval_level = 90, credibility_of_tolerance_interval = 80,
  covariate_cols = "age", covariate_values = 0.1, use_krishnamoorthy = TRUE
)
res_k$data
```

`calculate_ci` *Calculate Credibility Intervals*

Description

This function calculates credibility intervals for a specified column, grouped by another column.

Usage

```
calculate_ci(data, ColToAggregate, PercentForCI, ByColumn)
```

Arguments

`data`	A data frame or data.table containing the data
`ColToAggregate`	Column name to calculate credibility intervals for
`PercentForCI`	Credibility level (between 0 and 100)
`ByColumn`	Column name to group by

Value

A data frame with the following columns:

`By`	The grouping variable
`N`	Number of observations in each group
`CI_Lower`	Lower bound of the credibility interval
`CI_Upper`	Upper bound of the credibility interval
`Median`	Median value for each group

Examples

```
set.seed(1)
data <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  Value = c(rnorm(50, 10, 2), rnorm(50, 15, 3))
)
calculate_ci(data, "Value", 95, "Group")
```

`calculate_gamma_difference` *Calculate Gamma Difference for Krishnamoorthy Method*

Description

This function calculates the difference between the target gamma (confidence level) and the actual gamma achieved for a given set of points. For a candidate interval (g1, g2) = (x, 2 * gP_mean - x), the achieved gamma is the percentage of draws whose own interval lies inside it, that is with `lower_quantile` >= `g1` and `upper_quantile` <= `g2`.

Usage

```
calculate_gamma_difference(scatter_data, gP_mean, point_x_vector, gamma_actual)
```

Arguments

`scatter_data`	A data frame containing at least the following columns: • By: A grouping variable • lower_quantile: Lower quantile values • upper_quantile: Upper quantile values
`gP_mean`	Numeric, the gP_mean value for the By group
`point_x_vector`	Numeric, the x-coordinate(s) to evaluate
`gamma_actual`	Numeric, the target gamma (confidence level) percentage

Value

A list containing:

- gamma_differences: Differences between target and achieved gamma
- points_satisfying: Number of points satisfying the condition
- total_points: Total number of points evaluated

References

Krishnamoorthy K, Mathew T (2009). *Statistical Tolerance Regions: Theory, Applications, and Computation*. Wiley, Hoboken, NJ. doi:10.1002/9780470473900.

Examples

```
data <- data.frame(
  By = rep("A", 4),
  lower_quantile = c(-2, -1, -3, -2),
  upper_quantile = c(2, 3, 1, 2)
)
calculate_gamma_difference(data, gP_mean = 0.5, point_x_vector = c(-2, -1.5, -1),
                           gamma_actual = 95)
```

`calculate_gP_mean` *Calculate gP Mean for Krishnamoorthy Method*

Description

This function calculates the gP_mean value for each By group in the data, which is the average of the midpoints between lower and upper quantiles.

Usage

```
calculate_gP_mean(scatter_data, P)
```

Arguments

`scatter_data`	A data frame containing at least the following columns: • By: A grouping variable • lower_quantile: Lower quantile values • upper_quantile: Upper quantile values
`P`	Numeric, the probability level (not used directly but included for API consistency)

Value

A named vector of gP_mean values, with names corresponding to By values

References

Krishnamoorthy K, Mathew T (2009). *Statistical Tolerance Regions: Theory, Applications, and Computation*. Wiley, Hoboken, NJ. doi:10.1002/9780470473900.

Examples

```
data <- data.frame(
  By = c("A", "A", "B", "B"),
  lower_quantile = c(-2, -1, -3, -2),
  upper_quantile = c(2, 3, 1, 2)
)
calculate_gP_mean(data, 90)
```

`calculate_PI` *Calculate Prediction Intervals*

Description

Calculates Bayesian prediction intervals from MCMC draws. Every draw m defines a normal distribution with mean μ_m and variance σ_m^2. The posterior predictive distribution of a future observation is approximated by the average of these distribution functions,

$$\tilde{p}(x) = \frac{1}{M} \sum_{m=1}^{M} \Phi\left(\frac{x - \mu_m}{\sigma_m}\right),$$

and the limits of a two-sided $P\backslash\,(1-P)/2$ and $(1+P)/2$ quantiles of $\tilde{p}$. The quantiles are found numerically: a bisection that is accelerated by Newton steps whenever they stay inside the bracket. The bracket [smallest, largest] of the per-draw quantiles always contains the quantile of the average, so the sign of $\tilde{p}(x) - p$ is guaranteed to differ at the two starting points.

Usage

```
calculate_PI(data, percent_for_pi, Eta = 1e-08)
```

Arguments

`data`	A data frame or data.table containing at least the following columns: **By** A grouping variable **mean** The mean of the normal distribution of each draw **variance** The variance of the normal distribution of each draw
`percent_for_pi`	The desired prediction interval percentage (between 0 and 100)
`Eta`	Relative convergence tolerance: the search stops when the bracket, or a Newton step, is smaller than `Eta` times the standard deviation of the predictive distribution (default = 1e-8).

Value

A data frame with the following columns:

By The grouping variable

PI_Lower Lower bound of the prediction interval

PI_Upper Upper bound of the prediction interval

Examples

```
data <- data.frame(
  By = c("A", "A", "B", "B"),
  mean = c(10, 11, 20, 21),
  variance = c(2, 2.1, 3, 3.1)
)
calculate_PI(data, percent_for_pi = 95)
```

`calculate_predicted_distributions` *Calculate Predicted Distributions*

Description

This function calculates predicted distributions based on fixed effects, random parameters, and covariates. It computes means, medians, quantiles, and variances for each group in the data.

Usage

```
calculate_predicted_distributions(
  data,
  fixed_effects,
  random_params,
  by,
  tolerance_level,
  multiplication_factor = 1,
  covariate_cols = NULL,
  covariate_values = NULL,
  verbose = FALSE
)
```

Arguments

`data`	A data frame or data.table containing the data
`fixed_effects`	Character vector of column names for fixed effects
`random_params`	Character vector of column names for random parameters
`by`	Character, name of the column to group by. If "Single Chain Dataset", a SingleChain column will be created
`tolerance_level`	Numeric, tolerance level percentage (between 0 and 100)
`multiplication_factor`	Numeric, factor to multiply random parameters variance by (default = 1)
`covariate_cols`	Character vector of column names for covariates (default = NULL)
`covariate_values`	Numeric vector of values to multiply with covariates (default = NULL)
`verbose`	Logical, if TRUE, print detailed information during execution (default = FALSE)

Value

A data frame with the following columns:

`By`	The grouping variable
`mean`	Mean of fixed effects plus covariates
`median`	Median value
`lower_quantile`	Lower quantile based on tolerance level
`upper_quantile`	Upper quantile based on tolerance level
`variance`	Variance of random parameters

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
)

pred_dist <- calculate_predicted_distributions(
  mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_level = 90
)
head(pred_dist)

# With a covariate scaled by 0.1
calculate_predicted_distributions(
  mcmc, fixed_effects = "beta", random_params = "sigma2", by = "Group",
  tolerance_level = 90, covariate_cols = "age", covariate_values = 0.1
) |> head()
```

`calculate_tolerance_interval` *Calculate Tolerance Intervals*

Description

This function calculates tolerance intervals for specified lower and upper columns, grouped by another column. Tolerance intervals contain a specified proportion of the population with a given confidence level.

Usage

```
calculate_tolerance_interval(
  data,
  LowerColToAggregate,
  UpperColToAggregate,
  ConfidenceLevel,
  ByColumn
)
```

Arguments

data	A data frame or data.table containing the data
LowerColToAggregate	Column name for the lower bound values
UpperColToAggregate	Column name for the upper bound values
ConfidenceLevel	Confidence level for the tolerance interval (between 0 and 100)
ByColumn	Column name to group by

Value

A data frame with the following columns:

By	The grouping variable
TI_Lower	Lower bound of the tolerance interval
TI_Upper	Upper bound of the tolerance interval

Examples

```
set.seed(1)
data <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  lower_quantile = c(rnorm(50, 5, 1), rnorm(50, 10, 2)),
  upper_quantile = c(rnorm(50, 15, 1), rnorm(50, 20, 2))
)
calculate_tolerance_interval(data, "lower_quantile", "upper_quantile", 95, "Group")
```

---

concatenate_by_columns

Concatenate Multiple By Columns

Description

Creates a new column by concatenating values from multiple by columns with an underscore separator.

Usage

```
concatenate_by_columns(
  data,
  by_columns = NULL,
  new_column_name = "concatenated_by_column"
)
```

Arguments

data	A data frame containing the input data
by_columns	Character vector of column names to concatenate. If NULL, returns the original dataset.
new_column_name	Character, name for the concatenated column (default = "concatenated_by_column")

Value

A data frame with the original data and a new column containing concatenated by values

Examples

```
data <- data.frame(
  by1 = c(1, 1, 1, 2, 3, 3, 2),
  by2 = c("orange", "orange", "apple", "apple", "apple", "apple", "apple"),
  value = seq_len(7)
)
concatenate_by_columns(data, by_columns = c("by1", "by2"))
concatenate_by_columns(data, by_columns = c("by1", "by2"), new_column_name = "my_by_column")
```

---

create_all_cdf_dataframes

Create CDF Data for All By-Values

Description

This function creates CDF data for all by-values in the predicted distributions output.

Usage

```
create_all_cdf_dataframes(
  pred_dist,
  n_points = 20,
  ci_level = 90,
  log_normal = FALSE
)
```

Arguments

pred_dist	A dataframe containing predicted distributions (output from calculate_predicted_distributions)
n_points	Integer, number of points for CDF evaluation (default = 20)
ci_level	Numeric, confidence interval level as percentage (default = 90)
log_normal	Logical, whether to use log-normal transformation (default = FALSE)

Value

A list of dataframes, each containing CDF data for a specific by-value

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
)

pred_dist <- calculate_predicted_distributions(
  mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_level = 90
)
all_cdf_data <- create_all_cdf_dataframes(pred_dist)
names(all_cdf_data)
```

---

create_CDF *Create CDF Data for Posterior Predictive Distribution*

Description

Creates a dataframe or list of dataframes containing cumulative distribution function (CDF) data for posterior predictive distribution analysis. If by_value is specified, returns a single dataframe for that by-value. If by_value is NULL, returns a list of dataframes for all by-values.

Usage

```
create_CDF(
  data = NULL,
  fixed_effects = NULL,
  random_params = NULL,
  by = NULL,
  by_value = NULL,
  tolerance_level = NULL,
  n_points = 20,
```

```
  ci_level = 90,
  multiplication_factor = 1,
  log_normal = FALSE,
  covariate_cols = NULL,
  covariate_values = NULL,
  pred_dist = NULL
)
```

Arguments

data	A data frame containing the input data. If provided, predicted distributions will be calculated internally.
fixed_effects	Character vector of column names for fixed effects. Required if data is provided.
random_params	Character vector of column names for random parameters. Required if data is provided.
by	Character, name of the column to group by. If not provided and pred_dist is NULL, defaults to "Single Chain Dataset".
by_value	Character or numeric, the specific by-value to process. Use "Single Chain" for single chain dataset. If NULL, CDF data will be created for all by-values.
tolerance_level	Numeric, tolerance level percentage (between 0 and 100). Required if data is provided.
n_points	Integer, number of points for CDF evaluation (between 3 and 300, inclusive)
ci_level	Numeric, confidence interval level as percentage (default = 90)
multiplication_factor	Numeric, factor to multiply random parameters variance by (default = 1)
log_normal	Logical, whether to use log-normal transformation (default = FALSE)
covariate_cols	Character vector of column names for covariates (default = NULL)
covariate_values	Numeric vector of values to multiply with covariates (default = NULL)
pred_dist	Optional. A dataframe containing predicted distributions. If provided, data parameters are ignored.

Value

If by_value is specified: A dataframe containing CDF data for that by-value. If by_value is NULL: A list of dataframes, each containing CDF data for a different by-value.

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
```

```
)

cdf <- create_CDF(
  data = mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_level = 90, n_points = 20
)
names(cdf)
```

`create_cdf_dataframe` *Create CDF Data for Posterior Predictive Distribution*

Description

Creates a dataframe containing cumulative distribution function (CDF) data for posterior predictive distribution analysis.

Usage

```
create_cdf_dataframe(
  pred_dist,
  by_value,
  n_points = 20,
  ci_level = 90,
  log_normal = FALSE
)
```

Arguments

`pred_dist`	A dataframe containing predicted distributions (output from calculate_predicted_distributions)
`by_value`	Character or numeric, the specific by-value to process. Use "Single Chain" for single chain dataset
`n_points`	Integer, number of points for CDF evaluation (default = 20)
`ci_level`	Numeric, confidence interval level as percentage (default = 90)
`log_normal`	Logical, whether to use log-normal transformation (default = FALSE)

Value

A dataframe containing:

x Numeric vector of x values (transformed if log_normal = TRUE)

x_log Numeric vector of x values in log space

y_mean Numeric vector with the median, over the MCMC draws, of the distribution functions at each x value (the column name is kept for compatibility)

ci_lower Numeric vector of lower confidence interval bounds

ci_upper Numeric vector of upper confidence interval bounds

by_value Character or numeric, the by-value being processed

range_info List containing xmin, xmax in both original and log space

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
)

pred_dist <- calculate_predicted_distributions(
  mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_level = 90
)
cdf_data <- create_cdf_dataframe(pred_dist, by_value = "A")
head(cdf_data)
```

---

data_InputFileTester2 *Synthetic MCMC output for examples and tests*

Description

A small, entirely synthetic data set that mimics the output of an MCMC sampler for a linear mixed model. It contains no real study data.

Usage

```
data_InputFileTester2
```

Format

A data frame with 19 rows and 8 variables:

iteration MCMC iteration number

mu Intercept draw

t1, t2 Fixed-effect draws

v1, v2 Variance-component draws

By Numeric grouping variable (three groups)

Bytwo Character grouping variable

Source

Simulated.

`find_optimal_point` *Find Optimal Point for Krishnamoorthy Method*

Description

This function finds the optimal point (g1, g2) for the Krishnamoorthy method using a bisection algorithm. The optimal point satisfies the constraint g2 = 2*gP_mean - g1 and achieves the target gamma (confidence level).

Usage

```
find_optimal_point(
  scatter_data,
  gP_mean_vector,
  gamma_actual,
  max_iterations = 300,
  tolerance = 0.01
)
```

Arguments

`scatter_data`	A data frame containing at least the following columns: • By: A grouping variable • lower_quantile: Lower quantile values • upper_quantile: Upper quantile values
`gP_mean_vector`	Named vector of gP_mean values, with names corresponding to By values
`gamma_actual`	Numeric, the target gamma (confidence level) percentage
`max_iterations`	Integer, maximum number of iterations for bisection (default = 300)
`tolerance`	Numeric, convergence criterion for bisection (default = 0.01)

Value

A list of results, one element per By value, each containing:

- By: The By value
- g1: The optimal x-coordinate
- g2: The optimal y-coordinate
- gamma_difference: Difference between target and achieved gamma
- gamma_achieved: The achieved gamma percentage
- points_satisfying: Number of points satisfying the condition
- total_points: Total number of points evaluated
- iterations: Number of iterations used

References

Krishnamoorthy K, Mathew T (2009). *Statistical Tolerance Regions: Theory, Applications, and Computation*. Wiley, Hoboken, NJ. doi:10.1002/9780470473900.

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
)

pred_dist <- calculate_predicted_distributions(
  mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_level = 90
)
gP_mean <- calculate_gP_mean(pred_dist, 90)
optimal <- find_optimal_point(pred_dist, gP_mean, gamma_actual = 80)
optimal[["A"]]
```

---

format_intervals_table

Format Intervals Table with Double Header

Description

Creates a formatted DT table with two header levels showing statistical intervals in the same format as the Shiny app.

Usage

```
format_intervals_table(result, show_reducible = FALSE)
```

Arguments

`result`	Output list from perform_calculations() containing data and summary
`show_reducible`	Logical, whether to include reducible imprecision columns (default = FALSE)

Value

A DT::datatable object with formatted headers showing interval percentages

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
)

res <- perform_calculations(
  mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_level = 90
)
tbl <- format_intervals_table(res)
```

---

get_probability_table *Create Posterior Predictive Probability Table*

Description

Creates a table showing probability values and confidence intervals for a given x value or range using exactly the same calculations as the Shiny app.

Usage

```
get_probability_table(
  data = NULL,
  fixed_effects = NULL,
  random_params = NULL,
  by = "Single Chain Dataset",
  by_value = "1",
  x_value,
  tolerance_level = NULL,
  ci_percent = 90,
  multiplication_factor = 1,
  log_normal = FALSE,
  n_points = 100,
  pred_dist = NULL
)
```

Arguments

data	A data frame containing the input data. If provided, predicted distributions will be calculated internally.
fixed_effects	Character vector of column names for fixed effects. Required if data is provided.
random_params	Character vector of column names for random parameters. Required if data is provided.

by	Character, name of the column to group by. Required if data is provided. Default is "Single Chain Dataset".
by_value	Character or numeric, the specific by-value to process. Default is "1".
x_value	Character or numeric, single value or range in format "[min,max]"
tolerance_level	Numeric, tolerance level percentage (between 0 and 100). Required if data is provided.
ci_percent	Numeric, confidence interval percentage (default = 90)
multiplication_factor	Numeric, factor to multiply random parameters variance by (default = 1)
log_normal	Logical, whether to use log-normal transformation (default = FALSE)
n_points	Integer, number of points for CDF evaluation (default = 100)
pred_dist	Optional. A dataframe containing predicted distributions. If provided, data parameters are ignored.

Value

A data frame formatted as a table for display

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
)

get_probability_table(
  data = mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", by_value = "A", x_value = "10", tolerance_level = 90
)
```

---

load_n_inspect_data *Load and Inspect Data*

Description

This function attempts to load data using various methods and performs data quality checks. It tries different reading methods in order of efficiency and flexibility.

Usage

```
load_n_inspect_data(file_path = NULL, data = NULL, silent = FALSE)
```

Arguments

file_path	Character string specifying the path to the data file.
data	Optional data frame. If provided, only inspection is performed.
silent	Logical, if TRUE suppresses messages (but not warnings). Default is FALSE.

Value

A data frame containing the loaded data.

Examples

```
df <- data.frame(x = 1:10, y = letters[1:10])
data <- load_n_inspect_data(data = df, silent = TRUE)

# From a file
path <- tempfile(fileext = ".csv")
utils::write.csv(df, path, row.names = FALSE)
data <- load_n_inspect_data(file_path = path, silent = TRUE)
unlink(path)
```

---

lognorm_adjuster *Adjust Log-Normal Data and Calculate Reducible Imprecision*

Description

This function takes interval data and performs two main operations:

1. If log_normal is TRUE, exponentiates the values to transform from log space
2. Calculates reducible imprecision percentages for upper and lower bounds

Usage

```
lognorm_adjuster(IntervalsOutputInitial, log_normal)
```

Arguments

IntervalsOutputInitial	A data frame containing interval data with columns: Median, CI_Lower, CI_Upper, TI_Lower, TI_Upper, PI_Lower, PI_Upper
log_normal	Logical indicating whether to transform data from log space

Value

A data frame with the original columns plus:

ReducibleUpper	Upper reducible imprecision percentage
ReducibleLower	Lower reducible imprecision percentage

Examples

```
data <- data.frame(
  Median = 1, CI_Lower = 0.5, CI_Upper = 1.5,
  TI_Lower = 0.3, TI_Upper = 1.7, PI_Lower = 0.4, PI_Upper = 1.6
)
lognorm_adjuster(data, log_normal = TRUE)
```

---

parse_x_value_input *Parse X Value Input*

Description

Parses a string input representing either a single numeric value or a range [min,max].

Usage

```
parse_x_value_input(input_text)
```

Arguments

input_text	Character string to parse, either a single number or a range in the format "[min,max]"

Value

A list with components:

type Either "single" for a single value or "range" for a range

value The numeric value (if type is "single")

xmin The minimum value (if type is "range")

xmax The maximum value (if type is "range")

NULL if the input cannot be parsed

Examples

```
parse_x_value_input("5.2")
parse_x_value_input("[2.1, 7.8]")
```

`perform_calculations` *Perform Statistical Interval Calculations*

Description

This function coordinates the calculation of various statistical intervals including predicted distributions, confidence intervals, tolerance intervals, and prediction intervals.

Usage

```
perform_calculations(
  data,
  fixed_effects,
  random_params,
  by,
  tolerance_level,
  multiplication_factor = 1,
  percent_for_ci = 95,
  percent_for_pi = 95,
  confidence_of_tolerance = 95,
  log_normal = FALSE
)
```

Arguments

`data`	A data frame containing the input data
`fixed_effects`	Character vector of column names for fixed effects
`random_params`	Character vector of column names for random parameters
`by`	Character, name of the column to group by
`tolerance_level`	Numeric, tolerance level percentage (between 0 and 100)
`multiplication_factor`	Numeric, factor to multiply random parameters variance by (default = 1)
`percent_for_ci`	Numeric, percentage for confidence intervals (default = 95)
`percent_for_pi`	Numeric, percentage for prediction intervals (default = 95)
`confidence_of_tolerance`	Numeric, confidence level for tolerance intervals (default = 95)
`log_normal`	Logical, whether to transform results from log space (default = FALSE)

Value

A list containing:

- data: A data frame with calculated intervals and reducible imprecision metrics
- summary: A string summarizing the calculation parameters

Relationship to `bayesian_statistical_intervals()`

`bayesian_statistical_intervals` is the main function of the package. It supports covariates, the Krishnamoorthy two-sided tolerance intervals and returns the predicted distributions. `perform_calculations()` is kept for backward compatibility; new code should use `bayesian_statistical_intervals()`. Note that this function reports reducible imprecision as a percentage, whereas `bayesian_statistical_intervals()` reports it as a fraction.

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
)

res <- perform_calculations(
  mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_level = 90
)
res$data
```

`plot_intervals` *Create Interactive Statistical Intervals Plot*

Description

Creates an interactive plot showing Confidence Intervals (CI), Prediction Intervals (PI), and Tolerance Intervals (TI) using plotly. The plot matches the style and functionality of the original Shiny app and includes parameter information.

Usage

```
plot_intervals(
  result,
  show_reducible = FALSE,
  x_axis_title = NULL,
  show_summary = FALSE
)
```

Arguments

`result`	A list output from perform_calculations() containing data and summary
`show_reducible`	Logical, whether to show reducible imprecision ribbons (default = FALSE)
`x_axis_title`	Character, custom title for x-axis (optional)
`show_summary`	Logical, whether to print the summary in the console (default = FALSE)

Value

A plotly object displaying the statistical intervals

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
)

res <- perform_calculations(
  mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_level = 90
)
p <- plot_intervals(res)
p2 <- plot_intervals(res, show_summary = TRUE)
```

---

plot_krishnamoorthy *Create Krishnamoorthy Two-Sided Tolerance Interval Plot*

Description

Creates an interactive plot showing the Krishnamoorthy method results for two-sided tolerance intervals, including scatter points, optimal line, and optimal point.

Usage

```
plot_krishnamoorthy(
  data,
  by_value,
  optimal_point,
  tolerance_level,
  confidence_level,
  log_normal = FALSE
)
```

Arguments

data	A data frame containing lower and upper quantiles
by_value	Character or numeric, the specific by-value to plot
optimal_point	List containing Krishnamoorthy method results with elements: • g1: optimal lower bound • g2: optimal upper bound

- gamma_achieved: achieved confidence level

`tolerance_level`
: Numeric, tolerance level percentage (between 0 and 100)

`confidence_level`
: Numeric, confidence level percentage (between 0 and 100)

`log_normal`
: Logical, whether to transform results from log space (default = FALSE)

Value

A plotly object containing the Krishnamoorthy plot

References

Krishnamoorthy K, Mathew T (2009). *Statistical Tolerance Regions: Theory, Applications, and Computation*. Wiley, Hoboken, NJ. doi:10.1002/9780470473900.

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
)

pred_dist <- calculate_predicted_distributions(
  mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_level = 90
)
gP_mean <- calculate_gP_mean(pred_dist, 90)
optimal <- find_optimal_point(pred_dist, gP_mean, gamma_actual = 80)
p <- plot_krishnamoorthy(
  data = pred_dist, by_value = "A", optimal_point = optimal[["A"]],
  tolerance_level = 90, confidence_level = 80
)
```

---

plot_one_sided_intervals

Plot One-Sided Statistical Intervals

Description

Creates interactive plots showing one-sided statistical intervals (CI, PI, TI) with separate plots for upper and lower bounds.

Usage

```
plot_one_sided_intervals(
  data,
  show_reducible = FALSE,
  log_normal = FALSE,
  selected_by_values = NULL
)
```

Arguments

data	A data frame containing interval data with columns: By, Median, CI_Lower, CI_Upper, PI_Lower, PI_Upper, TI_Lower, TI_Upper
show_reducible	Logical, whether to show reducible imprecision ribbons (default = FALSE)
log_normal	Logical, whether values are in log-normal scale (default = FALSE)
selected_by_values	Optional vector of specific by values to include in the plot

Value

A list containing two plotly objects:

- upper: Plot for upper bounds
- lower: Plot for lower bounds

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
)

res <- bayesian_statistical_intervals(
  mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_interval_level = 90
)
plots <- plot_one_sided_intervals(res$data)
plots$upper
plots$lower
```

plot_posterior_predictive *Plot Posterior Predictive Distribution*

Description

Creates an interactive plot of the posterior predictive distribution using plotly. Its only a legacy function to keep compatibility with the older versions.

Usage

```
plot_posterior_predictive(
  cdf_data,
  x_value = NULL,
  title = NULL,
  show_ci = TRUE,
  line_colors = list(main = "rgba(0,100,80,1)", ci = "red", ci_ribbon =
    "rgba(255,0,0,0.3)", reference = "black"),
  show_reference_lines = TRUE
)
```

Arguments

cdf_data	A dataframe from create_cdf_dataframe() containing CDF information, or a list of such dataframes
x_value	Optional. Single numeric value or range [min,max] as character string (e.g., "[2,5]")
title	Optional. Custom plot title. If NULL, auto-generates based on by-value
show_ci	Logical. Whether to show confidence interval ribbon (default = TRUE)
line_colors	List of colors for different elements (optional)
show_reference_lines	Logical. Whether to show reference lines for x_value (default = TRUE)

Value

A plotly object containing the posterior predictive distribution plot

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
)
```

```
pred_dist <- calculate_predicted_distributions(
  mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_level = 90
)
cdf_data <- create_cdf_dataframe(pred_dist, by_value = "A")
p <- plot_posterior_predictive(cdf_data)
# Mark a single x-value, or a range
p_x <- plot_posterior_predictive(cdf_data, x_value = "10")
p_range <- plot_posterior_predictive(cdf_data, x_value = "[9, 11]")
```

`quantile_scatterplot` *Generate Interactive Quantile Scatterplot with Ellipses*

Description

This function creates an interactive scatterplot visualizing the relationship between lower and upper quantiles of posterior predictive distributions from MCMC samples, with confidence ellipses. It uses plotly for interactivity and can work with either raw data or pre-calculated distributions.

Usage

```
quantile_scatterplot(
  data = NULL,
  fixed_effects = NULL,
  random_params = NULL,
  by = "Single Chain Dataset",
  pred_dist = NULL,
  selected_by_values = NULL,
  tolerance_level = 95,
  ellipse_confidence = 95,
  multiplication_factor = 1,
  log_normal = FALSE,
  covariate_cols = NULL,
  covariate_values = NULL
)
```

Arguments

`data`	A data frame containing MCMC samples. Required if pred_dist is NULL.
`fixed_effects`	Character vector of column names for fixed effects. Required if pred_dist is NULL.
`random_params`	Character vector of column names for random parameters. Required if pred_dist is NULL.
`by`	Character string specifying the grouping variable. Default is "Single Chain Dataset".

`pred_dist` Optional pre-calculated predicted distributions from calculate_predicted_distributions(). If provided, data, fixed_effects, and random_params are ignored.

`selected_by_values` Optional vector of specific by values to include in the plot. If NULL, all by values are shown. Default is NULL.

`tolerance_level` Numeric value between 0 and 100 for tolerance interval percentage. Default is 95.

`ellipse_confidence` Numeric value between 0 and 100 for credible ellipses. Default is 95.

`multiplication_factor` Positive numeric value to adjust variance components. Default is 1.

`log_normal` Logical indicating whether to transform predictions to log-normal scale. Default is FALSE.

`covariate_cols` Character vector of column names for covariates. Default is NULL.

`covariate_values` Numeric vector of values to multiply with covariates. Default is NULL.

Details

The function creates a scatterplot where:

- Each point represents a single MCMC sample
- X-axis shows the upper quantile
- Y-axis shows the lower quantile
- Points are grouped by the specified 'by' variable
- Confidence ellipses show the concentration of points
- Diamond markers show the mean position for each group
- Interactive tooltips provide detailed information

Value

A plotly object containing the interactive quantile scatterplot with ellipses.

References

Krishnamoorthy K, Mathew T (2009). *Statistical Tolerance Regions: Theory, Applications, and Computation*. Wiley, Hoboken, NJ. doi:10.1002/9780470473900.

Examples

```
set.seed(1)
mcmc <- data.frame(
  Group = rep(c("A", "B"), each = 50),
  beta = c(rnorm(50, 10, 0.5), rnorm(50, 12, 0.5)),
  sigma2 = rgamma(100, shape = 20, rate = 10),
  age = rnorm(100, 50, 5)
```

```
)

# From raw MCMC draws
p1 <- quantile_scatterplot(
  data = mcmc, fixed_effects = "beta", random_params = "sigma2",
  by = "Group", tolerance_level = 95, ellipse_confidence = 90
)

# From pre-calculated distributions, with a covariate
pred_dist <- calculate_predicted_distributions(
  mcmc, "beta", "sigma2", "Group", tolerance_level = 95,
  covariate_cols = "age", covariate_values = 0.1
)
p2 <- quantile_scatterplot(pred_dist = pred_dist, tolerance_level = 95,
                           ellipse_confidence = 95)
```

`start_app_view` *Launch the Shiny Application*

Description

Starts the interactive application for uploading MCMC output and calculating and plotting statistical intervals. Uploads of up to 30 MB are accepted; the previous value of the `shiny.maxRequestSize` option is restored when the application is closed.

Usage

```
start_app_view(...)
```

Arguments

`...`	arguments to pass to shiny::runApp

Value

Called for its side effect of running the application; returns `NULL` invisibly when the application is closed.

Examples

```
if(interactive()){
  start_app_view()
}
```

Index

∗ **datasets**
data_InputFileTester2, 17

add_krishnamoorthy_ti, 3

bayesian_statistical_intervals, 4, *25*

calculate_ci, 6
calculate_gamma_difference, 7
calculate_gP_mean, 8
calculate_PI, 9
calculate_predicted_distributions, 10
calculate_tolerance_interval, 11
concatenate_by_columns, 12
create_all_cdf_dataframes, 13
create_CDF, 14
create_cdf_dataframe, 16

data_InputFileTester2, 17

find_optimal_point, 18
format_intervals_table, 19

get_probability_table, 20

load_n_inspect_data, 21
lognorm_adjuster, 22

parse_x_value_input, 23
perform_calculations, 24
plot_intervals, 25
plot_krishnamoorthy, 26
plot_one_sided_intervals, 27
plot_posterior_predictive, 29

quantile_scatterplot, 30

start_app_view, 32