Package 'BayesRTMB'

June 1, 2026

Type Package

Title Bayesian Inference Using 'RTMB'

Version 0.1.1

Description Provides tools for Markov chain Monte Carlo (MCMC) and Maximum A Posteriori (MAP) estimation utilizing the 'RTMB' package. It supports various statistical models including generalized linear mixed models, factor analysis, item response theory, and multidimensional unfolding. The package allows users to easily transition between frequentist and Bayesian paradigms using a unified interface. Automatic differentiation and Laplace approximation follow Kristensen et al. (2016) <doi:10.18637/jss.v070.i05>, and MCMC sampling uses the No-U-Turn Sampler described by Hoffman and Gelman (2014) <https://jmlr.org/papers/v15/hoffman14a.html>.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

LazyData true

Depends R (>= 4.1.0), RTMB

Imports R6, MASS

URL https://github.com/norimune/BayesRTMB, https://norimune.github.io/BayesRTMB/

BugReports https://github.com/norimune/BayesRTMB/issues

Suggests knitr, rmarkdown, GPArotation, future, future.apply, progressr

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

NeedsCompilation no

Author Hiroshi Shimizu [aut, cre]

Maintainer Hiroshi Shimizu <simizu706@gmail.com>

Repository CRAN

Date/Publication 2026-06-01 08:40:26 UTC

Contents

`ADVI_method` *Automatic Differentiation Variational Inference (ADVI)*

Description

Automatic Differentiation Variational Inference (ADVI)

Usage

```
ADVI_method(
  model,
  par_list,
  pl_full,
  iter = 3000,
  tol_rel_obj = 0.001,
  window_size = 100,
  num_samples = 1000,
  alpha = 0.01,
  laplace = FALSE,
  print_freq = 500,
  method = c("meanfield", "fullrank", "hybrid"),
  update_progress = NULL,
  update_interval = 100
)
```

Arguments

model	An RTMB objective function object (‘ad_obj‘).
par_list	A list defining the structure of parameters to be estimated.
pl_full	A list defining the full structure of parameters including random effects.
iter	Integer; fixed number of iterations for the optimization. Default is 3000.
tol_rel_obj	Numeric; relative tolerance for the ELBO to check convergence. Default is 0.001.
window_size	Integer; size of the moving window to calculate the median ELBO. Default is 100.
num_samples	Integer; number of posterior draws to generate after optimization. Default is 1000.
alpha	Numeric; learning rate (step size) for the Adam optimizer. Default is 0.01.
laplace	Logical; whether Laplace approximation is used. Default is FALSE.
print_freq	Integer; frequency of printing progress to the console. Set to 0 to disable. Default is 500.
method	Character; type of variational distribution. One of "meanfield", "fullrank", or "hybrid". Default is "meanfield".
update_progress	Optional function to update a progress bar.
update_interval	Integer; interval for updating the progress bar. Default is 100.

Value

A list containing ‘fit‘, ‘random_fit‘, ‘elbo_history‘, ‘elbo_final‘, ‘rel_obj_final‘, and ‘converged‘.

---

bayes_factor *Calculate Bayes Factor*

Description

Compare two models by calculating the Bayes Factor based on their marginal likelihoods.

Usage

```
bayes_factor(logml1, logml2, error_threshold = 0.2)
```

Arguments

logml1	The first model fit (e.g., ‘mcmc_fit‘) or its log-marginal likelihood.
logml2	The second model fit (e.g., ‘mcmc_fit‘) or its log-marginal likelihood.
error_threshold	Threshold for warning about high bridge-sampling error (default 0.2).

Value

An object of class 'bayes_factor' containing the Bayes factor, log Bayes factor, approximate estimation error, and interpretation.

Examples

```
# Compare two models using Bayes Factor
data(debate, package = "BayesRTMB")
fit1 <- rtmb_lm(sat ~ talk, data = debate)
fit2 <- rtmb_lm(sat ~ talk + perf, data = debate)
map1 <- fit1$optimize()
map2 <- fit2$optimize()
bf <- bayes_factor(map1, map2)
print(bf)
```

---

beverage *Beverage Preference Data*

Description

A dataset containing preference ratings for various beverages. This data was used to demonstrate a random effect model in metric multidimensional unfolding.

Usage

```
beverage
```

Format

A data frame (or matrix) with [INSERT NUMBER OF ROWS] rows and [INSERT NUMBER OF COLUMNS] columns.

AJ Apple Juice preference rating

BT Black Tea preference rating

CF Coffee preference rating

CL Cola preference rating

GT Green Tea preference rating

OJ Orange Juice preference rating

OT Oolong Tea preference rating

Source

Adachi, K. (2000). A Random Effect Model in Metric Multidimensional Unfolding. *The Japanese Journal of Behaviormetrics*, 27(1), 12-23. doi:10.2333/jbhmk.27.12

`BigFive` *Big Five Personality Traits Data*

Description

A dataset containing responses to 20 items measuring the Big Five personality traits. This dataset was originally collected by the package author. The items are arranged in a cyclical order representing the five factors: Extraversion, Neuroticism, Openness, Agreeableness, and Conscientiousness.

Usage

```
BigFive
```

Format

A data frame with 10 rows and 20 columns:

BF1 Extraversion item 1

BF2 Neuroticism item 1

BF3 Openness item 1

BF4 Agreeableness item 1

BF5 Conscientiousness item 1

BF6 Extraversion item 2

BF7 Neuroticism item 2

BF8 Openness item 2

BF9 Agreeableness item 2

BF10 Conscientiousness item 2

BF11 Extraversion item 3

BF12 Neuroticism item 3

BF13 Openness item 3

BF14 Agreeableness item 3

BF15 Conscientiousness item 3

BF16 Extraversion item 4

BF17 Neuroticism item 4

BF18 Openness item 4

BF19 Agreeableness item 4

BF20 Conscientiousness item 4

Source

Originally collected by the package author.

`Classic_Fit` *Classic fit object*

Description

An R6 class representing the results of a classical (frequentist) estimation.

Super class

`BayesRTMB::RTMB_Fit_Base` -> `Classic_Fit`

Public fields

`model` The 'RTMB_Model' object used for estimation.

`fit` The result of the estimation (dataframe or lm object).

`par` a named list of parameter estimates.

`vcov` Variance-covariance matrix of fixed effects.

`se_method` Character string specifying the method used for standard errors.

`cluster` Character string specifying the cluster variable name, if any.

`robust_type` Character string specifying the robust standard error correction type, if used.

`bootstrap_results` A matrix containing bootstrap samples, if applicable.

`test_results` List of additional test results (e.g., chisq.test).

`view` Character vector of parameter names to prioritize in summary.

`par_vec` Numeric vector of parameter estimates.

`objective` Final objective value.

`log_lik` Log-likelihood.

`restricted_log_lik` Restricted log-likelihood, if classical estimation used REML-style marginalization.

`rss` Residual sum of squares for classical linear models, if available.

`df_residual` Residual degrees of freedom for classical linear models, if available.

`convergence` Convergence code.

`sd_rep` TMB sdreport object.

`df_fixed` Dataframe of fixed effects results.

`random_effects` Dataframe of random effects results.

`df_transform` Dataframe of transformed parameters.

`df_generate` Dataframe of generated quantities.

`opt_history` Dataframe of optimization history.

`transform` List of transformed parameters.

`generate` List of generated quantities.

`se_samples` List of simulated samples for SE estimation.

`par_unc` Numeric vector of unconstrained parameter estimates.

`vcov_unc` Variance-covariance matrix of parameters in unconstrained space.

`ci_method` Method used for CI estimation.

`laplace` Whether Laplace approximation was used.

`map` Parameter mapping used.

`df_method` Character string specifying the degrees of freedom calculation method.

`idx_fix_active` Numeric vector; mapping between active parameters and full unconstrained vector.

`show_df` Logical; whether to display degrees of freedom in the summary output.

Methods

Public methods:

- `Classic_Fit$get_point_estimate()`
- `Classic_Fit$new()`
- `Classic_Fit$robust_se()`
- `Classic_Fit$compute_robust()`
- `Classic_Fit$bootstrap()`
- `Classic_Fit$compute_bootstrap()`
- `Classic_Fit$AIC()`
- `Classic_Fit$BIC()`
- `Classic_Fit$WAIC()`
- `Classic_Fit$diagnose()`
- `Classic_Fit$print()`
- `Classic_Fit$logLik()`
- `Classic_Fit$EAP()`
- `Classic_Fit$MAP()`
- `Classic_Fit$summary()`
- `Classic_Fit$anova()`
- `Classic_Fit$lsmeans()`
- `Classic_Fit$.calc_contrast()`
- `Classic_Fit$.get_lsmeans_df()`
- `Classic_Fit$.construct_par_list()`
- `Classic_Fit$clone()`

Method `get_point_estimate()`**:** Get point estimate for a target parameter.

Usage:

```
Classic_Fit$get_point_estimate(target, ...)
```

Arguments:

`target` Target parameter name.

`...` Additional arguments, ignored for classic fits.

Returns: Matrix, array, vector, or scalar point estimate.

Method `new()`**:** Create a new 'Classic_Fit' object.

Usage:

```
Classic_Fit$new(
  model,
  par_vec = NULL,
  par = NULL,
  objective = NULL,
  log_lik = NULL,
  restricted_log_lik = NULL,
  convergence = NULL,
  sd_rep = NULL,
  df_fixed = NULL,
  random_effects = NULL,
  df_transform = NULL,
  df_generate = NULL,
  opt_history = NULL,
  transform = NULL,
  generate = NULL,
  se_samples = NULL,
  par_unc = NULL,
  vcov_unc = NULL,
  ci_method = "wald",
  laplace = TRUE,
  map = NULL,
  test_results = list(),
  view = NULL,
  fit = NULL,
  vcov = NULL,
  df_method = "auto",
  idx_fix_active = NULL,
  show_df = TRUE,
  rss = NULL,
  df_residual = NULL,
  ...
)
```

Arguments:

`model` The 'RTMB_Model' object.

`par_vec` Numeric vector of parameter estimates.

`par` List of parameter estimates.

`objective` Final objective value.

`log_lik` Full log-likelihood used for information criteria.

`restricted_log_lik` Restricted log-likelihood, if classical estimation used REML-style marginalization.

`convergence` Convergence code.

`sd_rep` TMB sdreport object.

`df_fixed` Dataframe of fixed effects results.

`random_effects` Dataframe of random effects results.

`df_transform` Dataframe of transformed parameters.

`df_generate` Dataframe of generated quantities.

`opt_history` Dataframe of optimization history.

`transform` List of transformed parameters.

`generate` List of generated quantities.

`se_samples` List of simulated samples for SE estimation.

`par_unc` Parameter vector on unconstrained scale.

`vcov_unc` Covariance matrix on unconstrained scale.

`ci_method` Method used for CI estimation.

`laplace` Whether Laplace approximation was used.

`map` Parameter mapping used.

`test_results` List of additional test results (e.g., chisq.test).

`view` Character vector of parameter names to prioritize in summary.

`fit` Legacy argument for backward compatibility (maps to df_fixed).

`vcov` Variance-covariance matrix of parameters.

`df_method` Method for degrees of freedom calculation.

`idx_fix_active` Numeric vector; mapping between active parameters and full unconstrained vector.

`show_df` Logical; whether to display degrees of freedom in the summary output.

`rss` Residual sum of squares for classical linear models, if available.

`df_residual` Residual degrees of freedom for classical linear models, if available.

`...` Additional arguments passed to the constructor.

Method `robust_se()`**:** Compute robust standard errors (sandwich estimator).

Usage:

```
Classic_Fit$robust_se(
  cluster = NULL,
  type = c("HC3", "HC0", "HC1", "CR1", "CR0"),
  inplace = FALSE,
  ...
)
```

Arguments:

`cluster` Character; variable name for clustering.

`type` Character; "HC3" (default), "HC0", or "HC1" for non-clustered robust SE; "CR1" or "CR0" for clustered robust SE.

`inplace` Logical; if FALSE, return a robust-SE copy without modifying the original fit. If TRUE, update the object in place.

`...` Additional arguments.

Returns: Self.

Method `compute_robust()`**:** (Deprecated) Use robust_se() instead.

Usage:

```
Classic_Fit$compute_robust(...)
```

Arguments:

`...` Arguments passed to robust_se(), including inplace.

Returns: Self.

Method `bootstrap()`**:** Compute nonparametric bootstrap standard errors and confidence intervals. Currently implemented only for mediation models. Bootstrap refits mediation equations with base R lm.fit()/glm.fit() when possible, and falls back to RTMB refits for unsupported families.

Usage:

```
Classic_Fit$bootstrap(n_boot = 1000, seed = NULL, inplace = FALSE, ...)
```

Arguments:

`n_boot` Integer; number of bootstrap samples.

`seed` Optional integer random seed.

`inplace` Logical; if FALSE, return a bootstrap-SE copy without modifying the original fit. If TRUE, update the object in place.

`...` Additional arguments.

Returns: A Classic_Fit object.

Method `compute_bootstrap()`**:** Compute nonparametric bootstrap standard errors and confidence intervals. Currently implemented only for mediation models. Bootstrap refits mediation equations with base R lm.fit()/glm.fit() when possible, and falls back to RTMB refits for unsupported families.

Usage:

```
Classic_Fit$compute_bootstrap(n_boot = 1000, seed = NULL, inplace = FALSE, ...)
```

Arguments:

`n_boot` Integer; number of bootstrap samples.

`seed` Optional integer random seed.

`inplace` Logical; if FALSE, return a bootstrap-SE copy without modifying the original fit. If TRUE, update the object in place.

`...` Additional arguments.

Returns: A Classic_Fit object.

Method `AIC()`**:** Get the AIC of the fitted model.

Usage:

```
Classic_Fit$AIC()
```

Method `BIC()`**:** Get the BIC of the fitted model.

Usage:

```
Classic_Fit$BIC()
```

Method `WAIC()`**:** WAIC is not defined for classical fits.

Usage:

```
Classic_Fit$WAIC()
```

Method `diagnose()`**:** Run basic diagnostics for the classical fit.

Usage:

```
Classic_Fit$diagnose(...)
```

Arguments:

`...` Additional arguments passed to 'diagnose_classic_fit()'.

Returns: A 'diagnose_BayesRTMB' object.

Method `print()`**:** Print the fit results.

Usage:

```
Classic_Fit$print(...)
```

Arguments:

`...` Additional arguments passed to 'summary()'.

Method `logLik()`**:** Get the Log-Likelihood of the fitted model.

Usage:

```
Classic_Fit$logLik()
```

Method `EAP()`**:** EAP estimates are not available for Classic_Fit.

Usage:

```
Classic_Fit$EAP(...)
```

Arguments:

`...` Ignored.

Method `MAP()`**:** MAP estimates are not available for Classic_Fit.

Usage:

```
Classic_Fit$MAP(...)
```

Arguments:

`...` Ignored.

Method `summary()`**:** Display a summary of the estimation results.

Usage:

```
Classic_Fit$summary(view = NULL, digits = 5, max_rows = 10, ranef = FALSE)
```

Arguments:

`view` Character vector of parameter names to prioritize or filter by.

`digits` Number of digits to print for estimates.

`max_rows` Maximum number of rows to display in the coefficient table.

`ranef` Logical; whether to show random effects in the summary.

Method `anova()`**:** Perform ANOVA (Wald F-tests / Chisq-tests) on the fitted model.

Usage:

```
Classic_Fit$anova(method = c("reml", "ls"), type = 3)
```

Arguments:

`method` Character; "reml" (standard) or "ls" (experimental).

`type` Integer; Type of Sum of Squares (only Type III supported currently).

Returns: A data frame containing the ANOVA table.

Method `lsmeans()`**:** Calculate Least Squares Means (Marginal Means) and contrasts.

Usage:

```
Classic_Fit$lsmeans(
  specs = NULL,
  pairwise = FALSE,
  simple = NULL,
  adjust = "holm",
  protect = FALSE
)
```

Arguments:

`specs` Character vector of factors to calculate means for.

`pairwise` Logical; whether to perform pairwise comparisons.

`simple` Character vector of factors to hold constant for simple main effects.

`adjust` Character; p-value adjustment method (e.g., "bonferroni", "holm", "none").

`protect` Logical; whether to use hierarchical (protected) testing.

Returns: A data frame containing the marginal means or contrasts.

Method `.calc_contrast()`**:** (Internal) Calculate metrics for a contrast.

Usage:

```
Classic_Fit$.calc_contrast(L, specs)
```

Arguments:

`L` Contrast matrix.

`specs` Variable names for lsmeans.

Returns: A data frame with estimate, SE, df, t-value, and p-value.

Method `.get_lsmeans_df()`**:** (Internal) Get representative DF for lsmeans.

Usage:

```
Classic_Fit$.get_lsmeans_df(specs)
```

Arguments:

`specs` Variable names for lsmeans.

Returns: Degrees of freedom.

Method `.construct_par_list()`**:** (Internal) Construct a list of parameters from the fit.

Usage:

```
Classic_Fit$.construct_par_list(fit)
```

Arguments:

`fit` The fit result (dataframe or lm object).

Returns: A named list of parameters.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Classic_Fit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`conditional_effects` *Calculate Conditional Effects*

Description

Calculate and visualize the predicted values (marginal effects) of a variable, potentially conditional on the levels of another variable (interaction).

Usage

```
conditional_effects(fit, effect, prob = 0.95, sd_multiplier = 1, ...)
```

Arguments

`fit`	Model fit object (e.g., ‘mcmc_fit‘, ‘map_fit‘).
`effect`	Name of the variable to visualize (e.g., "X1" or "X1:X2").
`prob`	Probability for the credible/confidence interval (default is 0.95).
`sd_multiplier`	Numeric. Multiplier for standard deviation when splitting continuous moderators (default is 1).
`...`	Additional arguments.

Value

An object of class ‘ce_rtmb‘ containing the predicted values and their credible intervals.

Examples

```
data(debate, package = "BayesRTMB")
fit <- rtmb_lm(sat ~ talk * perf, data = debate)
map_fit <- fit$optimize()
ce <- conditional_effects(map_fit, effect = "talk:perf")
plot(ce)
summary(ce)
```

conditional_effects.mcmc_fit *Calculate conditional effects for MCMC fit objects*

Description

Calculate conditional effects for MCMC fit objects

Usage

```
## S3 method for class 'mcmc_fit'
conditional_effects(
  fit,
  effect,
  prob = 0.95,
  sd_multiplier = 1,
  resolution = 100,
  ...
)
```

Arguments

fit	An object of class ‘MCMC_Fit‘.
effect	Name of the explanatory variable to visualize (e.g., "X1" or "X1:X2").
prob	Probability for the credible/confidence interval (default is 0.95).
sd_multiplier	Numeric. Multiplier for standard deviation when splitting continuous moderators (default is 1).
resolution	Grid resolution to calculate for continuous variables (default is 100).
...	Additional arguments.

Value

A ‘ce_rtmb‘ object.

debate *Debate Simulation Data*

Description

A simulated dataset containing individual satisfaction scores and related variables from a 3-person group debate experiment. This dummy data was created by the package author to demonstrate hierarchical (multilevel) modeling.

Usage

```
debate
```

Format

A data frame with 300 rows and 6 columns:

group Group identifier. Each group consists of 3 members.

sat Self-reported satisfaction level of the individual after the debate.

talk Amount of talk or utterances by the individual during the debate.

perf Evaluated performance of the group debate (group-level variable).

skill Evaluated conversation skill of the individual.

cond Experimental condition indicator (e.g., 0 = control, 1 = treatment).

Source

Simulated dummy data created by the package author.

Dim *Define parameter dimensions and types*

Description

Define parameter dimensions and types

Usage

```
Dim(dim = 1, type = NULL, lower = NULL, upper = NULL, random = FALSE)
```

Arguments

`dim`	Dimensions of the parameter.
`type`	Type of the parameter.
`lower`	Lower bound.
`upper`	Upper bound.
`random`	Logical; whether it is a random effect.

`distance` *Euclidean distance*

Description

Euclidean distance

Usage

```
distance(x, y, eps = 1e-08)
```

Arguments

`x`	A numeric vector.
`y`	A numeric vector.
`eps`	A small value added for numerical stability (default is 1e-8).

Value

The Euclidean distance between x and y.

`distributions` *Probability Distributions for RTMB Models*

Description

This page summarizes the probability distributions available within the 'rtmb_code' block. These functions are designed to be used with the Stan-like sampling syntax ('~'), which internally adds the log-density to the model's total log-posterior.

Details

Syntax Styles: In 'rtmb_code', you can specify distributions in two ways:

- **Sampling Syntax (Recommended):** `y ~ normal(mu, sigma)`
- **Explicit Function Call:** `lp <- lp + normal_lpdf(y, mu, sigma)`

Continuous Distributions (LPDF):

- `normal(mean, sd)`: Normal distribution.
- `lognormal(meanlog, sdlog)`: Lognormal distribution.
- `exponential(rate)`: Exponential distribution.
- `cauchy(location, scale)`: Cauchy distribution.
- `student_t(df, mu, sigma)`: Student's t-distribution.

- gamma(shape, rate): Gamma distribution.
- inverse_gamma(shape, scale): Inverse-gamma distribution.
- beta(a, b): Beta distribution.

Discrete Distributions (LPMF):

- bernoulli(prob) / bernoulli_logit(eta): Binary outcomes.
- binomial(size, prob) / binomial_logit(size, eta): Binomial outcomes.
- poisson(mean): Poisson count data.
- neg_binomial_2(mu, size): Negative binomial (mean/dispersion parameterization).
- ordered_logistic(eta, cutpoints): Ordered categorical outcomes.
- sequential_logistic(eta, cutpoints): Sequential ordered categorical outcomes.

Multivariate and Matrix Distributions:

- multi_normal(mean, Sigma): Standard multivariate normal distribution.
- multi_student_t(df, mean, Sigma): Multivariate Student's t-distribution.
- multi_cauchy(mean, Sigma): Multivariate Cauchy distribution (Student-t with df=1).
- lkj_corr(eta): LKJ prior for correlation matrices.
- dirichlet(alpha): Dirichlet distribution for simplexes.
- lower_tri_normal(mean, sd): Normal distribution for elements of a lower-triangular matrix.
- centered_tri_multi_normal(sigma): Multivariate normal for centered triangular matrices (used in identification constraints).
- sufficient_multi_normal_fa(S_mat, N, y_bar, mu, psi, Lambda): Factor analysis likelihood using sufficient statistics (highly efficient for large sample sizes).

Vectorization: Most univariate distributions are vectorized. If y and mu are vectors, y ~ normal(mu, sigma) will calculate the sum of log-densities for all elements efficiently.

---

ess_basic *Basic Effective Sample Size for a single chain or pooled chains*

Description

Basic Effective Sample Size for a single chain or pooled chains

Usage

```
ess_basic(sims)
```

Arguments

sims	A numeric vector or matrix of samples.

Value

A numeric value.

ess_bulk *Calculate Bulk Effective Sample Size*

Description

Calculate Bulk Effective Sample Size

Usage

```
ess_bulk(sims)
```

Arguments

sims	A matrix of samples (iterations x chains).

Value

A numeric value.

ess_tail95 *Calculate Tail Effective Sample Size (at 2.5% and 97.5% quantiles)*

Description

Calculate Tail Effective Sample Size (at 2.5% and 97.5% quantiles)

Usage

```
ess_tail95(sims)
```

Arguments

sims	A matrix of samples (iterations x chains).

Value

A numeric value.

fabs *Smooth absolute value function*

Description

Smooth absolute value function

Usage

```
fabs(x, eps = 1e-08)
```

Arguments

x	A numeric vector.
eps	A small value added for numerical stability (default is 1e-8).

Value

The smooth absolute value of x.

gaussian_process_lpdf *Gaussian Process Log-Density (Squared Exponential Kernel)*

Description

Calculates the log-density of a Gaussian Process with a Squared Exponential (RBF) kernel.

Usage

```
gaussian_process_lpdf(
  y,
  x,
  mean = 0,
  magnitude = 1,
  smoothing = 1,
  noise = 0.01,
  sum = TRUE
)
```

Arguments

y	Observation vector (N).
x	Coordinate vector or matrix (N x D).
mean	Mean vector (scalar or length N).
magnitude	Signal standard deviation (alpha).
smoothing	Length-scale (rho).
noise	Measurement noise standard deviation (sigma).
sum	Logical; whether to return the sum of log-densities.

Value

Log-density value.

---

generate_random_init *Generate Random Initial Values*

Description

Generates random initial values for model parameters by drawing from a uniform distribution on the unconstrained scale and then transforming them to the constrained scale.

Usage

```
generate_random_init(pl_full, par_list, range = 2)
```

Arguments

pl_full	A list containing the full parameter structure, including the ‘init‘ field which serves as a template.
par_list	A list defining the structure and constraints of the parameters, including ‘unc_length‘.
range	Numeric; the range for the uniform distribution ‘[-range, range]‘ used for generating unconstrained values. Default is 2.

Value

A numeric vector of initial values where ‘NA‘ elements are replaced with randomly generated constrained values.

inv_logit *Inverse logit function*

Description

Inverse logit function

Usage

```
inv_logit(x)
```

Arguments

x	A numeric vector.

Value

The inverse logit of x.

item_curve *Calculate Item Response Curve / Category Response Curve*

Description

Calculate Item Response Curve / Category Response Curve

Usage

```
item_curve(x, ...)
```

Arguments

x	An object of class
...	Additional arguments.

item_curve.RTMB_Fit_Base *Item Response Curve for RTMB_Fit_Base*

Description

Item Response Curve for RTMB_Fit_Base

Usage

```
## S3 method for class 'RTMB_Fit_Base'
item_curve(x, theta_seq = seq(-4, 4, length.out = 100), items = NULL, ...)
```

Arguments

x	An object of class RTMB_Fit_Base.
theta_seq	Sequence of trait values (ability) to evaluate.
items	Index or item names to restrict the calculation to specific items (optional).
...	Additional arguments.

Examples

```
fit <- rtmb_irt(data = BigFive[, 1:5], model = "2PL", type = "ordered")
map_fit <- fit$optimize()
ic <- item_curve(map_fit)
plot(ic)
```

item_info *Calculate Item Information Function*

Description

Calculate Item Information Function

Usage

```
item_info(x, ...)
```

Arguments

x	An object of class RTMB_Fit_Base
...	Additional arguments.

Examples

```
fit <- rtmb_irt(data = BigFive[, 1:5], model = "2PL", type = "ordered")
map_fit <- fit$optimize()
ii <- item_info(map_fit)
plot(ii)
```

`item_info.RTMB_Fit_Base` *Item Information Function for RTMB_Fit_Base*

Description

Item Information Function for RTMB_Fit_Base

Usage

```
## S3 method for class 'RTMB_Fit_Base'
item_info(x, theta_seq = seq(-4, 4, length.out = 100), items = NULL, ...)
```

Arguments

`x`	An object of class RTMB_Fit_Base.
`theta_seq`	Sequence of trait values (ability) to evaluate.
`items`	Index or item names to restrict the calculation to specific items (optional).
`...`	Additional arguments.

`log1m` *Log of one minus x*

Description

Log of one minus x

Usage

```
log1m(x)
```

Arguments

`x`	A numeric vector.

Value

The natural logarithm of 1 - x.

log1m_exp *Log of one minus exponential of x*

Description

Log of one minus exponential of x

Usage

```
log1m_exp(x)
```

Arguments

x	A numeric vector.

Value

The natural logarithm of 1 - exp(x).

log1p_exp *Log of one plus exponential of x*

Description

Log of one plus exponential of x

Usage

```
log1p_exp(x)
```

Arguments

x	A numeric vector.

Value

The natural logarithm of 1 + exp(x).

logit *Logit function*

Description

Logit function

Usage

```
logit(x)
```

Arguments

x	A numeric vector of probabilities.

Value

The logit of x.

log_det_chol *Log determinant of a Cholesky factor*

Description

Log determinant of a Cholesky factor

Usage

```
log_det_chol(L)
```

Arguments

L	A lower triangular Cholesky factor matrix.

Value

The log determinant of the corresponding covariance matrix.

log_mix *Log mixture of two probabilities*

Description

Log mixture of two probabilities

Usage

```
log_mix(theta, lp1, lp2)
```

Arguments

theta	Mixing proportion (between 0 and 1).
lp1	Log-probability of the first component.
lp2	Log-probability of the second component.

Value

The log-probability of the mixture.

log_softmax *Log-softmax function*

Description

Log-softmax function

Usage

```
log_softmax(x)
```

Arguments

x	A numeric vector.

Value

A numeric vector of log-softmax probabilities.

log_sum_exp *Log-sum-exp function*

Description

Log-sum-exp function

Usage

```
log_sum_exp(x, y = NULL)
```

Arguments

x	A numeric vector or scalar.
y	An optional numeric scalar.

Value

The log of the sum of the exponentials.

log_sum_exp_matrix *Log-sum-exp function for matrices (row-wise)*

Description

Log-sum-exp function for matrices (row-wise)

Usage

```
log_sum_exp_matrix(M)
```

Arguments

M	A numeric matrix or advector matrix.

Value

A numeric vector of row-wise log-sum-exp.

lsmeans *Least Squares Means (Marginal Means)*

Description

Calculate least squares means (also known as marginal means or predicted means) for categorical factors in a fitted model. It also supports pairwise comparisons and simple main effects.

Usage

```
lsmeans(object, specs, ...)
```

Arguments

object	A fitted model object (e.g., 'Classic_Fit').
specs	A character vector of factor names to calculate means for.
...	Additional arguments passed to the method.

make_bw_from_ydif *Make Best and Worst Responses from Best-Worst Pair Indices*

Description

Converts a matrix of Best-Worst pair indices ('Y_dif') into separate 'Best' and 'Worst' data frames. The returned values are positions within each row of 'sets', not the item labels stored in 'sets'.

Usage

```
make_bw_from_ydif(Y_dif, sets)

restore_bw_from_ydif(Y_dif, sets)
```

Arguments

Y_dif	Matrix or data frame of pair indices (N persons x P tasks). Each value must be an integer from 1 to 'C * (C - 1)', where 'C' is the number of items per task.
sets	Matrix or data frame of presented item sets (P tasks x C items).

Value

A list with two data frames: 'Best' and 'Worst'.

`make_glmer_re_terms` *Prepare GLMM Formula Components*

Description

Builds the response, fixed-effect model matrix, and lme4-style random-effect design components used by ‘rtmb_glmer()‘. This is the user-facing helper that reproduces what the wrapper places in the generated ‘setup‘ block.

Usage

```
make_glmer_re_terms(
  formula,
  data,
  family = "gaussian",
  resid_group = NULL,
  resid_time = NULL,
  within = NULL,
  factors = NULL,
  missing = "listwise"
)
```

Arguments

`formula`	lme4-style formula.
`data`	Data frame.
`family`	Character string of the distribution family.
`resid_group`	Optional residual-correlation grouping variable.
`resid_time`	Optional residual-correlation time variable.
`within`	Optional list for wide-to-long conversion when the response is written as ‘cbind(...)‘.
`factors`	Optional character vector of variables to treat as factors.
`missing`	Missing value handling strategy: "listwise".

Value

A list containing ‘Y‘, ‘X‘, ‘trials‘, ‘offset‘, ‘N‘, fixed-effect metadata, and random-effect terms.

make_glmer_Z_matrix *Reconstruct an Observation-Level Random-Effect Design Matrix*

Description

Converts the transposed block random-effect design matrix used internally by ‘rtmb_glmer()‘ into an observation-level matrix. This is kept as a small utility for users who want to work from the block matrix returned by ‘make_glmer_re_terms()‘.

Usage

```
make_glmer_Z_matrix(Zt, group_idx, N = length(group_idx))
```

Arguments

Zt	Transposed block random-effect design matrix.
group_idx	Integer group index for each observation.
N	Number of observations. Defaults to ‘length(group_idx)‘.

Value

A matrix with ‘N‘ rows and one column per random-effect coefficient.

make_init_mdu *Create Initial Values for Multidimensional Unfolding*

Description

Create Initial Values for Multidimensional Unfolding

Usage

```
make_init_mdu(
  Y,
  D,
  distance = c("squared", "euclidean"),
  alpha = c("random", "fix"),
  distance_eps = 1e-04,
  min_sigma = 0.1
)
```

Arguments

Y	Numeric matrix or data frame (N rows x M items).
D	Number of unfolding dimensions.
distance	Character; ‘"squared"‘ or ‘"euclidean"‘.
alpha	Character; ‘"random"‘ for item-specific alpha initial values or ‘"fix"‘ for a common alpha initial value.
distance_eps	Small positive constant added to the distance.
min_sigma	Minimum initial residual standard deviation.

Value

A named list of initial values.

---

make_ydif_from_bw *Make Best-Worst Pair Indices from Best and Worst Responses*

Description

Converts separate ‘Best‘ and ‘Worst‘ response matrices into ‘Y_dif‘ pair indices for Best-Worst MDU models. The returned index is the position of the ordered ‘(best, worst)‘ pair among all ‘C * (C - 1)‘ position pairs generated from each row of ‘sets‘.

Usage

```
make_ydif_from_bw(Best, Worst, sets)
```

Arguments

Best	Matrix or data frame of best responses (N persons x P tasks). Values must be positions within the corresponding row of ‘sets‘, from ‘1‘ to ‘ncol(sets)‘.
Worst	Matrix or data frame of worst responses (N persons x P tasks). Values must be positions within the corresponding row of ‘sets‘, from ‘1‘ to ‘ncol(sets)‘.
sets	Matrix or data frame of presented item sets (P tasks x C items).

Value

An integer matrix of pair indices (‘Y_dif‘).

map_est *Maximum A Posteriori (MAP) Estimate*

Description

Maximum A Posteriori (MAP) Estimate

Usage

```
map_est(z)
```

Arguments

z	A numeric vector of samples.

Value

A numeric value.

MAP_Fit *MAP fit object*

Description

MAP fit object

MAP fit object

Details

An R6 class storing optimization results from maximum a posteriori (MAP) estimation.

Super class

BayesRTMB::RTMB_Fit_Base -> map_fit

Public fields

model The 'RTMB_Model' object used for estimation.

par_vec Parameter vector on the unconstrained scale (constrained values unlisted).

par Parameter list on the constrained scale.

par_unc Parameter vector on the unconstrained scale (raw unconstrained values).

ci_method Method used for CI estimation ("wald", "sampling", or "none").

objective RTMB objective function object.

log_ml Log marginal likelihood or related model criterion.

`convergence` Optimizer convergence code.

`sd_rep` Standard deviation report object.

`df_fixed` Summary table for fixed-effect parameters.

`random_effects` Random effect estimates.

`df_transform` Summary table for transformed parameter estimates.

`df_generate` Summary table for generated quantity estimates.

`opt_history` A vector of optimize objective history.

`transform` List of transformed parameters maintaining their original dimensions.

`generate` List of generated quantities maintaining their original dimensions.

`se_samples` List of simulated samples for standard error estimation.

`laplace` Logical; whether Laplace approximation was used.

`vcov_unc` Variance-covariance matrix of parameters in unconstrained space.

`map` List; the parameter mapping used.

`marginal_vars` Character vector of parameter names requested through 'optimize(marginal = ...)'.

`laplace_random_vars` Character vector of all parameter names passed to 'MakeADFun(random = ...)' during Laplace approximation.

`idx_fix_active` Numeric vector; mapping between active parameters and full unconstrained vector.

`show_df` Logical; whether to display degrees of freedom in the summary output.

`view` Character vector of parameter names to prioritize in summary.

`fallback_needed` Logical; whether Hessian/SE fallback was used during optimization.

Methods

Public methods:

- `MAP_Fit$get_point_estimate()`
- `MAP_Fit$new()`
- `MAP_Fit$ranef()`
- `MAP_Fit$draws()`
- `MAP_Fit$summary()`
- `MAP_Fit$print()`
- `MAP_Fit$generated_quantities()`
- `MAP_Fit$WAIC()`
- `MAP_Fit$diagnose()`
- `MAP_Fit$profile()`
- `MAP_Fit$clone()`

Method `get_point_estimate()`**:** Get point estimate for a target parameter.

Usage:

```
MAP_Fit$get_point_estimate(target, ...)
```

Arguments:

`target` Target parameter name.

`...` Additional arguments, ignored for MAP fits.

Returns: Matrix, array, vector, or scalar point estimate.

Method `new()`**:** Create a new 'MAP_Fit' object.

Usage:

```
MAP_Fit$new(
  model,
  par_vec = NULL,
  par = NULL,
  objective = NULL,
  log_ml = NULL,
  convergence = NULL,
  sd_rep = NULL,
  df_fixed = NULL,
  random_effects = NULL,
  df_transform = NULL,
  df_generate = NULL,
  opt_history = NULL,
  transform = NULL,
  generate = NULL,
  se_samples = NULL,
  par_unc = NULL,
  ci_method = "wald",
  laplace = TRUE,
  map = NULL,
  vcov_unc = NULL,
  marginal_vars = NULL,
  laplace_random_vars = NULL,
  idx_fix_active = NULL,
  show_df = TRUE,
  view = NULL,
  fallback_needed = NULL
)
```

Arguments:

`model` The 'RTMB_Model' object used for estimation.

`par_vec` Parameter vector on the unconstrained scale (constrained values unlisted).

`par` Parameter list on the constrained scale.

`objective` The objective function value at the optimum.

`log_ml` Log marginal likelihood.

`convergence` Optimizer convergence code.

`sd_rep` The 'sdreport' object from TMB.

`df_fixed` Data frame of fixed effects estimates and CIs.

`random_effects` Data frame of random effects estimates and CIs.

`df_transform` Data frame of transformed parameters.

`df_generate` Data frame of generated quantities.

`opt_history` Data frame of optimization history.

`transform` List of transformed parameters maintaining their original dimensions.

`generate` List of generated quantities maintaining their original dimensions.

`se_samples` List of simulated samples for standard error estimation.

`par_unc` Parameter vector on the unconstrained scale (raw values).

`ci_method` Method used for CI estimation ("wald", "sampling", or "none").

`laplace` Logical; whether Laplace approximation was used.

`map` List; the parameter mapping used.

`vcov_unc` Variance-covariance matrix of parameters in unconstrained space.

`marginal_vars` Character vector of parameter names requested through 'optimize(marginal = ...)'.

`laplace_random_vars` Character vector of all parameter names passed to 'MakeADFun(random = ...)' during Laplace approximation.

`idx_fix_active` Numeric vector; mapping between active parameters and full unconstrained vector.

`show_df` Logical; whether to display degrees of freedom in the summary output. Default is TRUE.

`view` Character vector of parameter names to prioritize in summary.

`fallback_needed` Logical; whether Hessian/SE fallback was used during optimization.

Method `ranef()`**:** Return random effect estimates as a named list.

Usage:

```
MAP_Fit$ranef()
```

Returns: A named list of random effect estimates.

Method `draws()`**:** Extract samples from the asymptotic posterior distribution.

Usage:

```
MAP_Fit$draws(
  pars = NULL,
  inc_random = FALSE,
  inc_transform = TRUE,
  inc_generate = TRUE,
  ...
)
```

Arguments:

`pars` Character or numeric vector specifying the names or indices of parameters to extract.

`inc_random` Logical; whether to include random effects.

`inc_transform` Logical; whether to include transformed parameters.

`inc_generate` Logical; whether to include generated quantities.

`...` Ignored.

Returns: An array of samples [iterations, 1, parameters].

Method `summary()`**:** Summarize MAP estimates.

Usage:

```
MAP_Fit$summary(
  pars = NULL,
  max_rows = 10,
  digits = 5,
  ranef = FALSE,
  view = NULL
)
```

Arguments:

`pars` Character vector specifying the names of parameters to summarize. If NULL, all available parameters are summarized.

`max_rows` Maximum number of rows to print in summaries. Default is 10.

`digits` Number of digits to print.

`ranef` Logical; whether to also display random effect estimates. Default is FALSE.

`view` Character vector of parameter names to prioritize or filter by.

Returns: A summary object, typically a data frame.

Method `print()`**:** Print a brief summary of the fitted object.

Usage:

```
MAP_Fit$print(pars = NULL, max_rows = 10, digits = 5, ...)
```

Arguments:

`pars` Character vector specifying the names of parameters to summarize.

`max_rows` Maximum number of rows to print in summaries.

`digits` Number of digits to print.

`...` Additional arguments passed to the ‘summary‘ method.

Returns: The object itself, invisibly.

Method `generated_quantities()`**:** Compute generated quantities from the MAP estimate.

Usage:

```
MAP_Fit$generated_quantities(code)
```

Arguments:

`code` An ‘rtmb_code({ ... })‘ or ‘{ ... }‘ block containing the logic to be calculated using the MAP estimate.

Returns: The ‘MAP_Fit‘ object itself (invisibly). Results are added or updated in the ‘generate‘ list and ‘df_generate‘.

Method `WAIC()`**:** Compute an approximate WAIC from sampling-based uncertainty propagation.

Usage:

```
MAP_Fit$WAIC()
```

Returns: A ‘waic_BayesRTMB‘ object.

Method `diagnose()`**:** Run basic diagnostics for the MAP fit.

Usage:

```
MAP_Fit$diagnose(...)
```

Arguments:

`...` Additional arguments passed to ‘diagnose_map_fit()‘.

Returns: A ‘diagnose_BayesRTMB‘ object.

Method `profile()`**:** Calculate Profile Likelihood confidence intervals for specific parameters.

Usage:

```
MAP_Fit$profile(
  pars = NULL,
  level = 0.95,
  trace = FALSE,
  digits = 5,
  show_plot = FALSE,
  quiet = FALSE,
  jacobian = "none",
  ...
)
```

Arguments:

`pars` Character vector of parameter names to profile. If NULL, all fixed parameters are profiled.

`level` Confidence level (default is 0.95).

`trace` Logical; whether to print profiling progress. Default is FALSE.

`digits` Integer; number of decimal places to print. Default is 5.

`show_plot` Logical; whether to plot the profile likelihood curves. Default is FALSE.

`quiet` Logical; whether to suppress text output. Default is FALSE.

`jacobian` Character; "none" (default), "random", or "all". Whether to include Jacobian adjustments for transformations.

`...` Additional arguments passed to TMB::tmbprofile (e.g., ytol).

Returns: A data frame containing the profile-based confidence intervals, with the raw profile objects stored in the "profiles" attribute.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MAP_Fit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`math_functions` *Mathematical and Matrix Utility Functions for RTMB Models*

Description

This page summarizes the utility functions provided to assist in writing efficient and numerically stable model code within ‘rtmb_code‘. These functions are specifically designed to be compatible with RTMB’s Automatic Differentiation (AD).

Details

1. Link and Inverse Functions:

- `logit(x)`: Computes the logit transformation `log(x/(1-x))`.
- `inv_logit(x)`: Computes the inverse logit (logistic) transformation `1/(1+exp(-x))`.

2. Numerical Stability Utilities: These functions use specialized algorithms to prevent overflow/underflow in AD calculations.

- `log_sum_exp(x)`: Safely computes `log(sum(exp(x)))` using the log-sum-exp trick.
- `log1p_exp(x)`: Computes `log(1 + exp(x))` stably.
- `log1m_exp(x)`: Computes `log(1 - exp(x))` stably for `x < 0`.
- `log_softmax(x)`: Computes the log of the softmax function.
- `fabs(x)`: A smooth version of the absolute value function `abs(x)` to ensure differentiability at zero.

3. Matrix and Vector Transformations: Used to convert unconstrained vectors into structured matrices (e.g., for factor analysis or identification constraints).

- `sum_to_zero(x)`: Transforms a vector of length K-1 into a vector of length K that sums to zero.
- `to_lower_tri(x, M, D)`: Fills a matrix of size M x D with elements from vector `x` in a lower-triangular fashion.
- `to_centered_matrix(x, R, C)`: Creates an R x C matrix where each column sums to zero.
- `to_centered_tri(x, R, C)`: Creates an R x C matrix with column-wise sum-to-zero constraints on the lower elements (useful for identification in factor analysis).

4. Linear Algebra for AD:

- `log_det_chol(L)`: Calculates the log-determinant of a covariance matrix from its Cholesky factor `L`.
- `quad_form_chol(x, L)`: Computes the quadratic form x^T Sigma^(-1) x using the Cholesky factor `L`.
- `distance(x, y)`: Computes the Euclidean distance between two vectors with a small epsilon for stability.

`MCMC_Fit` *MCMC fit object*

Description

MCMC fit object

MCMC fit object

Details

An R6 class storing posterior samples and related information from MCMC estimation.

Bayes factors are computed as ratios of marginal likelihoods estimated by bridge sampling. The comparison model can be constructed by fixing parameters with 'fixed = list(...)', or supplied as an already fitted 'MCMC_Fit' object via 'comparison_fit'. Bayes factors are computed only by marginal-likelihood model comparison.

Super class

`BayesRTMB::RTMB_Fit_Base` -> `mcmc_fit`

Public fields

`model` An 'RTMB_Model' object used for estimation.

`fit` Posterior draws for model parameters.

`random_fit` Posterior draws for random effects.

`transform_fit` Posterior draws for transformed parameters.

`transform_dims` Dimension information for transformed parameters.

`generate_fit` Posterior draws for generated quantities.

`generate_dims` Dimension information for generated quantities.

`eps` Step size used by the sampler.

`accept` Acceptance statistics from sampling.

`treedepth` Tree depth used in HMC/NUTS sampling.

`laplace` Logical; whether Laplace approximation was used.

`posterior_mean` Posterior mean estimates.

`log_ml` Numeric value storing the calculated log marginal likelihood from bridge sampling.

`comparison_fit` An `MCMC_Fit` object containing the fitted comparison model.

`max_treedepth` Maximum tree depth requested for NUTS/HMC.

`pd_error_count` Positive-definite/singularity errors treated as lp = -Inf by chain.

Methods

Public methods:

- `MCMC_Fit$get_point_estimate()`
- `MCMC_Fit$new()`
- `MCMC_Fit$print()`
- `MCMC_Fit$draws()`
- `MCMC_Fit$summary()`
- `MCMC_Fit$unconstrain_draws()`
- `MCMC_Fit$log_prob()`
- `MCMC_Fit$bridgesampling()`
- `MCMC_Fit$WAIC()`

- MCMC_Fit$diagnose()
- MCMC_Fit$bayes_factor()
- MCMC_Fit$transformed_draws()
- MCMC_Fit$generated_quantities()
- MCMC_Fit$resolve_switching()
- MCMC_Fit$clone()

Method `get_point_estimate()`**:** Get point estimate for a target parameter.

Usage:

```
MCMC_Fit$get_point_estimate(target, chains = NULL, best_chains = NULL)
```

Arguments:

`target` Target parameter name.

`chains` Numeric vector of chains to include. If NULL, all chains are used.

`best_chains` Integer; number of best chains to retain based on mean log-posterior.

Returns: Matrix, array, vector, or scalar point estimate.

Method `new()`**:** Create a new 'MCMC_Fit' object.

Usage:

```
MCMC_Fit$new(
  model,
  fit,
  random_fit,
  eps,
  accept,
  treedepth,
  laplace,
  posterior_mean,
  max_treedepth = NULL,
  pd_error_count = NULL
)
```

Arguments:

`model` An 'RTMB_Model' object used for estimation.

`fit` Posterior draws for model parameters.

`random_fit` Posterior draws for random effects, if available.

`eps` Step size used by the sampler.

`accept` Acceptance statistics from sampling.

`treedepth` Tree depth used in HMC/NUTS sampling.

`laplace` Logical; whether Laplace approximation was used.

`posterior_mean` Posterior mean estimates.

`max_treedepth` Maximum tree depth requested for NUTS/HMC.

`pd_error_count` Positive-definite/singularity errors treated as 'lp = -Inf' by chain.

Method `print()`**:** Print a brief summary of the fitted object.

Usage:

```
MCMC_Fit$print(...)
```

Arguments:

`...` Additional arguments.

Returns: The object itself, invisibly.

Method `draws()`**:** Extract posterior draws for selected parameters.

Usage:

```
MCMC_Fit$draws(
  pars = NULL,
  chains = NULL,
  best_chains = NULL,
  inc_random = FALSE,
  inc_transform = TRUE,
  inc_generate = TRUE
)
```

Arguments:

`pars` Character or numeric vector specifying the names or indices of parameters to extract. If NULL, all available parameters are extracted.

`chains` Numeric vector specifying the chains to extract. If NULL, draws from all chains are returned.

`best_chains` Integer; number of best chains to retain based on mean log-posterior (lp).

`inc_random` Logical; whether to include random effects in the output. Default is FALSE.

`inc_transform` Logical; whether to include transformed parameters in the output. Default is TRUE.

`inc_generate` Logical; whether to include generated quantities in the output. Default is TRUE.

Returns: Posterior draws.

Method `summary()`**:** Summarize posterior draws.

Usage:

```
MCMC_Fit$summary(
  pars = NULL,
  chains = NULL,
  best_chains = NULL,
  max_rows = 10,
  digits = 2,
  inc_random = FALSE,
  inc_transform = TRUE,
  inc_generate = TRUE
)
```

Arguments:

`pars` Character or numeric vector specifying the names or indices of parameters to summarize. If NULL, all available parameters are summarized.

`chains` Numeric vector specifying the chains to extract. If NULL, draws from all chains are used.

`best_chains` Integer; number of best chains to retain based on mean log-posterior (lp).

`max_rows` Integer; maximum number of rows to print in the summary table. Default is 10.

`digits` Integer; number of decimal places to print. Default is 2.

`inc_random` Logical; whether to include random effects in the summary. Default is FALSE.

`inc_transform` Logical; whether to include transformed parameters in the summary. Default is TRUE.

`inc_generate` Logical; whether to include generated quantities in the summary. Default is TRUE.

Returns: A summary object.

Method `unconstrain_draws()`**:** Transform posterior draws to the unconstrained scale.

Usage:

```
MCMC_Fit$unconstrain_draws()
```

Returns: Posterior draws on the unconstrained scale.

Method `log_prob()`**:** Evaluate log-probability values.

Usage:

```
MCMC_Fit$log_prob(safe = FALSE)
```

Arguments:

`safe` Logical; whether to wrap the evaluation in a tryCatch block. Default is FALSE for speed.

Returns: Numeric vector of log-probability values.

Method `bridgesampling()`**:** Estimate the marginal likelihood by bridge sampling.

Usage:

```
MCMC_Fit$bridgesampling(
  method = "normal",
  use_neff = TRUE,
  seed = NULL,
  max_iter = 100
)
```

Arguments:

`method` Character; the method to use for bridge sampling (e.g., "warp3", "normal"). Default is "warp3".

`use_neff` Logical; whether to use the effective sample size (ESS) to adjust for autocorrelation. Default is TRUE.

`seed` Integer; random seed for reproducibility. Default is NULL.

`max_iter` Integer; maximum number of iterations for the estimation algorithm. Default is 100.

Returns: Bridge sampling result.

Method `WAIC()`**:** Compute WAIC from pointwise generated log likelihood.

Usage:

```
MCMC_Fit$WAIC(...)
```

Arguments:

... Additional arguments passed to 'draws()', such as 'chains' or 'best_chains'.

Returns: A 'waic_BayesRTMB' object.

Method `diagnose()`**:** Run basic diagnostics for the MCMC fit.

Usage:

```
MCMC_Fit$diagnose(...)
```

Arguments:

... Additional arguments passed to 'diagnose_mcmc_fit()'.

Returns: A 'diagnose_BayesRTMB' object.

Method `bayes_factor()`**:** Calculate the Bayes factor by marginal-likelihood model comparison.

Usage:

```
MCMC_Fit$bayes_factor(
  fixed = NULL,
  comparison_fit = NULL,
  bs_method = "normal",
  error_threshold = 0.2,
  ...
)
```

Arguments:

`fixed` Named list of parameter values used to construct the comparison model. For example, `fixed = list(delta = 0)` or `fixed = list("b[x]" = 0)`.

`comparison_fit` Optional 'MCMC_Fit' object for an already fitted comparison model.

`bs_method` Character; the method to use for bridge sampling ("normal" or "warp3").

`error_threshold` Numeric; threshold for the approximate error warning.

... Additional arguments passed to 'sample()' when fitting the comparison model (e.g., `chains = 4`, `sampling = 4000`).

Returns: A list of class `bayes_factor_rtmb` containing Bayes factor results.

Method `transformed_draws()`**:** Compute transformed parameters from posterior draws.

Usage:

```
MCMC_Fit$transformed_draws(tran_fn = NULL)
```

Arguments:

`tran_fn` A function for transformed parameters.

Returns: Transformed parameter draws.

Method `generated_quantities()`**:** Compute generated quantities from posterior draws.

Usage:

```
MCMC_Fit$generated_quantities(code)
```

Arguments:

`code` An 'rtmb_code({ ... })' or '{ ... }' block containing the logic to be calculated using posterior samples.

Returns: The ‘MCMC_Fit‘ object itself (invisibly). Results are appended to the ‘generate_fit‘ field.

Method `resolve_switching()`**:** Resolve label switching in posterior draws.

Usage:

```
MCMC_Fit$resolve_switching(
  target,
  linked = NULL,
  overwrite = TRUE,
  scalar_fns = list()
)
```

Arguments:

`target` Character string specifying the target variable to base the relabeling on.

`linked` Character vector of variable names to be relabeled in the same order as the target. Default is NULL.

`overwrite` Logical; whether to overwrite the stored draws in the current object. Default is TRUE.

`scalar_fns` A named list of functions to apply to scalar variables for relabeling. Default is an empty list.

Returns: Relabeled draws or updated object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MCMC_Fit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`model_code` *Model Code Wrapper for RTMB*

Description

Model Code Wrapper for RTMB

Usage

```
model_code(expr, env = parent.frame())
```

Arguments

`expr`	A block of code containing model description.
`env`	Environment to assign to the generated function.

Value

A standard R function object taking (dat, par).

parameters_code *Code block for parameter definitions*

Description

Defines the list of parameters to be passed to 'rtmb_model'. By writing in the format 'variable_name = Dim(...)' within a block enclosed in '', evaluation is delayed, enabling strict error checking during model construction.

Usage

```
parameters_code(expr)
```

Arguments

expr	A parameter definition expression enclosed in ''.

Value

A lazily evaluated list of parameters.

parameter_types *Parameter Types and Constraints in RTMB Models*

Description

When declaring parameters in the `parameters` block using `Dim()`, you can specify a `type` to automatically apply structural constraints. These constraints ensure that parameters remain within their valid mathematical space during estimation.

Details

Standard Types:

- `"vector"`, `"matrix"`, `"array"`: Standard unconstrained containers. Use `lower` and `upper` for element-wise bounds.

Ordering Constraints:

- `"ordered"`: A vector where $x_1 < x_2 < \ldots < x_K$.
- `"positive_ordered"`: A vector where $0 < x_1 < x_2 < \ldots < x_K$.

Constrained Vectors:

- `"simplex"`: A vector where all elements are >= 0 and sum(x) = 1.
- `"centered"`: A vector of length K where sum(x) = 0. (Estimated using K-1 degrees of freedom).

Correlation and Covariance Matrices:

- `"corr_matrix"`: A symmetric, positive-definite correlation matrix (diagonal elements are 1).
- `"cov_matrix"`: A symmetric, positive-definite covariance matrix.
- `"CF_corr"`: The Cholesky factor of a correlation matrix.
- `"CF_cov"`: The Cholesky factor of a covariance matrix (diagonal elements are positive).

Special Structural Matrices:

- `"lower_tri"`: A lower-triangular matrix.
- `"positive_lower_tri"`: A lower-triangular matrix with positive diagonal elements.
- `"centered_matrix"`: A matrix where each column sums to zero.
- `"centered_tri"`: A triangular matrix with column-wise sum-to-zero constraints (often used for identification in factor analysis).

See Also

`rtmb_code`, `math_functions`, `distributions`

`plot.ce_rtmb` *Plot method for ce_rtmb class (Base R)*

Description

Plot method for ce_rtmb class (Base R)

Usage

```
## S3 method for class 'ce_rtmb'
plot(x, ...)
```

Arguments

`x`	An object of class ce_rtmb
`...`	Additional arguments.

plot.rtmb_lsmeans *Plot marginal means with error bars*

Description

Plot method for rtmb_lsmeans objects.

Usage

```
## S3 method for class 'rtmb_lsmeans'
plot(
  x,
  y,
  error_bar = "se",
  col = NULL,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  ...
)
```

Arguments

x	An rtmb_lsmeans object.
y	Ignored.
error_bar	Type of error bar ("se" or "ci").
col	Color of the bars.
main	Plot title.
xlab	X axis label.
ylab	Y axis label.
...	Additional arguments passed to barplot.

plot_acf *Plot autocorrelation for one variable across chains*

Description

Draw autocorrelation plots for a selected parameter across all chains.

Usage

```
plot_acf(x, var_idx = 1)
```

Arguments

x	A 3D array of posterior samples with dimensions ‘(iterations, chains, variables)‘.
var_idx	Integer. Index of the variable to plot.

Value

No return value. This function is called for its side effect of plotting.

---

plot_conditional_effects *Plot conditional effects*

Description

Calculate and plot conditional effects for a fitted model.

Usage

```
plot_conditional_effects(fit, effect, ...)
```

Arguments

fit	A fitted model object (e.g., ‘MCMC_Fit‘, ‘MAP_Fit‘, ‘VB_Fit‘).
effect	Name of the variable to visualize (e.g., "X1" or "X1:X2").
...	Additional arguments passed to conditional_effects or the plot method.

Value

Invisibly returns the plotted object of class ce_rtmb.

---

plot_dens *Plot posterior densities for MCMC samples*

Description

Draw kernel density plots for each parameter across chains.

Usage

```
plot_dens(x, mono = FALSE)
```

Arguments

x	A 3D array of posterior samples with dimensions ‘(iterations, chains, variables)‘. A 2D matrix ‘(iterations, chains)‘ is also allowed and is treated as a single variable.
mono	Logical. If ‘TRUE‘, plot in monochrome. If ‘FALSE‘, use blue shades.

Value

No return value. This function is called for its side effect of plotting.

---

plot_forest *Plot parameter estimates and credible intervals (Forest Plot)*

Description

Plot parameter estimates and credible intervals (Forest Plot)

Usage

```
plot_forest(
  x,
  prob = 0.95,
  point_estimate = c("median", "EAP", "mean", "MAP", "marginal_map", "joint_map")
)
```

Arguments

x	A 3D array of posterior samples with dimensions ‘(iterations, chains, variables)‘.
prob	Numeric. Probability mass for the credible interval (default is 0.95).
point_estimate	Character. Point estimate shown as dots. One of ‘"median"‘ (default), ‘"EAP"‘/‘"mean"‘, ‘"MAP"‘/‘"marginal_map"‘, or ‘"joint_map"‘. ‘"joint_map"‘ uses the draw with the largest ‘lp‘ variable.

Value

No return value.

plot_item_curve *Plot item/category response curves*

Description

Calculate and plot item/category response curves for a fitted IRT model.

Usage

```
plot_item_curve(fit, ...)
```

Arguments

fit	A fitted IRT model object.
...	Additional arguments passed to item_curve or the plot method.

Value

Invisibly returns the plotted object of class rtmb_item_curve.

plot_item_info *Plot item information functions*

Description

Calculate and plot item information functions for a fitted IRT model.

Usage

```
plot_item_info(fit, ...)
```

Arguments

fit	A fitted IRT model object.
...	Additional arguments passed to item_info or the plot method.

Value

Invisibly returns the plotted object of class rtmb_item_info.

plot_lsmeans *Plot least-squares marginal means*

Description

Calculate and plot marginal means for a fitted model.

Usage

```
plot_lsmeans(fit, specs, ...)
```

Arguments

`fit`	A fitted model object (e.g. 'Classic_Fit').
`specs`	Character vector specifying the variables for which marginal means are calculated.
`...`	Additional arguments passed to `lsmeans` or the plot method.

Value

Invisibly returns the plotted object.

plot_mdu *Plot Multidimensional Unfolding Configuration*

Description

Draws an item-person map for multidimensional unfolding models. Item locations are shown as labels, optional blue circles show item alpha/radius values, and the respondent locations are summarized by a two- dimensional kernel density contour.

Usage

```
plot_mdu(
  delta,
  theta = NULL,
  item_alpha = NULL,
  phi = NULL,
  dims = c(1, 2),
  radius = NULL,
  signs = c(1, 1),
  item_labels = NULL,
  show_phi = TRUE,
  show_density = TRUE,
  circle_scale = 1,
```

```
  alpha = 0.2,
  contour_n = 60,
  distance = c("auto", "squared", "euclidean"),
  point_estimate = c("EAP", "MAP", "mean", "marginal_map", "joint_map"),
  main = "MDU Configuration",
  xlab = NULL,
  ylab = NULL,
  ...
)
```

Arguments

`delta`	Item coordinate matrix (M items x D dimensions), or a fitted object with an ‘EAP()‘ method.
`theta`	Person coordinate matrix (N persons x D dimensions). If ‘delta‘ is a fitted object and ‘theta‘ is ‘NULL‘, it is extracted from the fit.
`item_alpha`	Optional item alpha vector used for item radii. If ‘delta‘ is a fitted object and ‘item_alpha‘ is ‘NULL‘, ‘alpha‘ is extracted when available.
`phi`	Deprecated alias for ‘item_alpha‘.
`dims`	Integer vector of length 2 specifying dimensions to plot.
`radius`	Numeric plot radius. If ‘NULL‘, a radius is chosen from the coordinates.
`signs`	Numeric vector of length 2. Use ‘-1‘ to flip an axis.
`item_labels`	Optional item labels. Defaults to row names or item numbers.
`show_phi`	Logical; whether to draw circles based on item alpha.
`show_density`	Logical; whether to draw density contours for ‘theta‘.
`circle_scale`	Numeric multiplier for item radii.
`alpha`	Circle transparency.
`contour_n`	Grid size passed to ‘MASS::kde2d()‘.
`distance`	Character; ‘"auto"‘, ‘"squared"‘, or ‘"euclidean"‘. Used to transform item alpha values into plotted radii. ‘"auto"‘ uses the fit’s stored distance when available.
`point_estimate`	Character; point estimate used when ‘delta‘ is a fit object. Passed to ‘estimate()‘.
`main`, `xlab`, `ylab`	Plot title and axis labels.
`...`	Additional arguments passed to ‘plot()‘.

Value

Invisibly returns a list with plotted ‘delta‘, ‘theta‘, and ‘item_alpha‘.

plot_pairs *Plot pairs for posterior samples*

Description

Draw a scatterplot matrix to examine posterior correlations between parameters.

Usage

```
plot_pairs(x, pars = NULL)
```

Arguments

x	A 3D array of posterior samples with dimensions '(iterations, chains, variables)'.
pars	Character vector or integer vector specifying which parameters to plot. If NULL (default), up to the first 10 parameters are plotted to prevent overplotting.

Value

No return value.

plot_test_info *Plot test information function*

Description

Calculate and plot test information function for a fitted IRT model.

Usage

```
plot_test_info(fit, ...)
```

Arguments

fit	A fitted IRT model object.
...	Additional arguments passed to test_info or the plot method.

Value

Invisibly returns the plotted object of class rtmb_test_info.

plot_trace *Plot MCMC trace plots*

Description

Draw trace plots for each parameter across chains.

Usage

```
plot_trace(x, mono = FALSE)
```

Arguments

x	A 3D array of posterior samples with dimensions ‘(iterations, chains, variables)‘. A 2D matrix ‘(iterations, chains)‘ is also allowed and is treated as a single variable.
mono	Logical. If ‘TRUE‘, plot in monochrome. If ‘FALSE‘, use blue shades.

Value

No return value. This function is called for its side effect of plotting.

print.bayes_factor *Print method for bayes_factor objects*

Description

Print method for bayes_factor objects

Usage

```
## S3 method for class 'bayes_factor'
print(x, digits = 4, ...)
```

Arguments

x	An object of class bayes_factor.
digits	Number of decimal places to display (default is 4).
...	Additional arguments.

print.bayes_factor_rtmb *Print method for bayes_factor_rtmb objects*

Description

Print method for bayes_factor_rtmb objects

Usage

```
## S3 method for class 'bayes_factor_rtmb'
print(x, digits = 4, ...)
```

Arguments

x	An object of class bayes_factor_rtmb.
digits	Number of decimal places to display (default is 4).
...	Additional arguments.

print.ce_rtmb *Print method for ce_rtmb class (automatically calls plot)*

Description

Print method for ce_rtmb class (automatically calls plot)

Usage

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

Arguments

x	An object of class ce_rtmb
...	Additional arguments.

print.ce_simple *Print simple effects*

Description

Print simple effects

Usage

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

Arguments

x	A ‘ce_simple‘ object.
digits	Number of digits to print.
...	Additional arguments.

print.summary_BayesRTMB *print for summary_BayesRTMB class*

Description

print for summary_BayesRTMB class

Usage

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

Arguments

x	An object of class summary_BayesRTMB
digits	integer
...	Additional arguments.

prior_flat *Specify a flat prior*

Description

‘prior_flat()‘ specifies that no additional prior density is added by the wrapper. This is the only prior type allowed for ‘classic()‘.

Usage

```
prior_flat()
```

Value

A list with class ‘"rtmb_prior"‘ and ‘type = "flat"‘.

prior_jzs *Specify a JZS (Jeffrey-Zellner-Siow) prior for t-tests*

Description

Specify a JZS (Jeffrey-Zellner-Siow) prior for t-tests

Usage

```
prior_jzs(r = 0.707)
```

Arguments

r	Numeric; scale parameter for the Cauchy prior on effect size delta. Default is 0.707.

Value

A list with class "rtmb_prior"

prior_normal *Specify normal/exponential priors for MAP and Bayesian inference*

Description

‘prior_normal()‘ specifies normal priors for location parameters and exponential priors for scale parameters. It is intended for MAP and Bayesian inference, not for ‘classic()‘.

Usage

```
prior_normal(
  Intercept_sd = 10,
  mu_sd = 10,
  b_sd = 10,
  sigma_rate = 5,
  tau_rate = 5,
  ...
)
```

Arguments

Intercept_sd	Standard deviation for the intercept prior. If ‘NULL‘, no intercept prior is added.
mu_sd	Standard deviation for mean/intercept priors. If ‘NULL‘, no mean prior is added.
b_sd	Standard deviation for coefficient priors. If ‘NULL‘, no coefficient prior is added.
sigma_rate	Rate for residual standard deviation priors. If ‘NULL‘, no sigma prior is added.
tau_rate	Rate for random-effect standard deviation priors. If ‘NULL‘, no tau prior is added.
...	Optional wrapper-specific hyperparameters.

Value

A list with class ‘"rtmb_prior"‘ and ‘type = "normal"‘.

prior_rhs *Specify a Regularized Horseshoe prior for continuous shrinkage*

Description

Specify a Regularized Horseshoe prior for continuous shrinkage

Usage

```
prior_rhs(expected_vars = 3, slab_scale = 2, slab_df = 4, ...)
```

Arguments

expected_vars	Expected number of non-zero variables. Default is 3.
slab_scale	Scale parameter for the slab distribution. Default is 2.0.
slab_df	Degrees of freedom for the slab distribution. Default is 4.0.
...	Optional hyperparameters for other distributions.

Value

A list with class "rtmb_prior"

---

prior_ssp *Specify a Spike-and-Slab prior for variable selection*

Description

Specify a Spike-and-Slab prior for variable selection

Usage

```
prior_ssp(ssp_ratio = 0.25, max_beta = 1, ...)
```

Arguments

ssp_ratio	Prior probability of inclusion for each variable. Default is 0.25.
max_beta	Maximum expected effect size. Default is 1.0.
...	Optional hyperparameters for other distributions.

Value

A list with class "rtmb_prior"

---

prior_uniform *Specify a flat prior*

Description

Deprecated alias of ‘prior_flat()‘.

Usage

```
prior_uniform()
```

Value

A list with class ‘"rtmb_prior"‘ and ‘type = "flat"‘.

prior_weak *Specify a weakly informative prior*

Description

Specify a weakly informative prior

Usage

```
prior_weak(sd_ratio = 0.5, max_beta = 1, ...)
```

Arguments

sd_ratio	Ratio of the prior standard deviation to the half-range of the response variable. Default is 0.5.
max_beta	Maximum expected effect size. Default is 1.0.
...	Optional hyperparameters for other distributions.

Value

A list with class "rtmb_prior"

quad_form_chol *Quadratic form using a Cholesky factor*

Description

Quadratic form using a Cholesky factor

Usage

```
quad_form_chol(x, L)
```

Arguments

x	A numeric vector.
L	A lower triangular Cholesky factor matrix.

Value

The value of the quadratic form.

quad_form_diag *Quadratic form with a diagonal matrix*

Description

Quadratic form with a diagonal matrix

Usage

```
quad_form_diag(m, v)
```

Arguments

m	A numeric matrix.
v	A numeric vector representing the diagonal elements.

Value

The value of the quadratic form.

quantile95 *Calculate 95% Quantiles*

Description

Calculate 95% Quantiles

Usage

```
quantile95(x)
```

Arguments

x	A numeric vector.

Value

A numeric vector of length 2.

read_mcmc_csv *Restore MCMC Fit from CSV*

Description

Restore MCMC Fit from CSV

Usage

```
read_mcmc_csv(model, name, dir = "BayesRTMB_mcmc", chains = 4, laplace = FALSE)
```

Arguments

model	An RTMB_Model object.
name	Base name of the saved CSVs.
dir	Directory where CSVs are saved. Default is "BayesRTMB_mcmc".
chains	Number of chains. Default is 4.
laplace	Logical; whether Laplace approximation was used. Default is FALSE.

Value

An MCMC_Fit object.

rtmb_code *Define an RTMB Model with Stan-like Syntax*

Description

rtmb_code is the primary interface for defining Bayesian or Frequentist models within the RTMB framework. It allows you to organize model logic into functional blocks while utilizing a Stan-like ~ operator for specifying priors and likelihoods.

Usage

```
rtmb_code(...)
```

Arguments

...	Model definition blocks.

Details

Key Blocks and Logic:

- `setup`: Pre-compilation data processing.
- `parameters`: Declaration of estimated parameters. See `parameter_types` for available constraints.
- `transform`: Definition of intermediate variables. Use `math_functions` for stable AD calculations.
- `model`: Likelihood and Priors. Refer to `distributions` for available sampling statements.
- `generate`: Post-hoc calculation of predictions or diagnostics.

Automatic Differentiation (AD) Note: All code within `parameters`, `transform`, and `model` is automatically differentiated by RTMB. To ensure numerical stability, use provided utility functions like `log_sum_exp` or `inv_logit` instead of raw algebraic implementations.

Value

A list of unevaluated code blocks.

See Also

`rtmb_model` for building the model object. `parameter_types` for constraining parameters. `distributions` for likelihood functions. `math_functions` for numerical utilities.

Examples

```
# Simulate data for a linear regression
set.seed(123)
N <- 100
x <- rnorm(N)
y <- 2.0 + 1.5 * x + rnorm(N, mean = 0, sd = 1)
data_list <- list(N = N, x = x, y = y)

# Define the model using rtmb_code
code <- rtmb_code(
  setup = {
    # Center the predictor variable (executed once)
    x_centered <- x - mean(x)
  },
  parameters = {
    # Define parameters and their constraints
    alpha = Dim(1)
    beta  = Dim(1)
    sigma = Dim(1, lower = 0)
  },
  transform = {
    # Calculate the linear predictor
    mu <- alpha + beta * x_centered
  },
  model = {
```

```
    # Priors
    alpha ~ normal(0, 10)
    beta  ~ normal(0, 10)
    sigma ~ exponential(1)

    # Likelihood (Vectorized)
    y ~ normal(mu, sigma)
  },
  generate = {
    # Calculate generated quantities
    y_pred <- mu

    # Must return a named list
    list(y_pred = y_pred)
  }
)

# Create the model object
mod <- rtmb_model(data = data_list, code = code)

# Fit the model using MAP estimation
map_res <- mod$optimize()
map_res$summary(pars = c("alpha", "beta", "sigma"))

# The generated quantity 'y_pred' can also be summarized
map_res$summary("y_pred", max_rows = 5)
```

`rtmb_corr` *Fit a Correlation Model using RTMB*

Description

'rtmb_corr' fits a correlation model to estimate means, standard deviations, and correlation structures. It supports simple correlation, multilevel correlation, and classical frequentist estimation.

Usage

```
rtmb_corr(
  x = NULL,
  data = NULL,
  ID = NULL,
  covariates = NULL,
  method = c("pearson", "spearman", "reml"),
  prior = prior_flat(),
  y_range = NULL,
  init = NULL,
  fixed = NULL,
  missing = c("listwise", "fiml", "pairwise"),
```

```
  WAIC = FALSE
)
```

Arguments

x	A matrix, data frame, formula, or expression (e.g., cbind(V1, V2)) of response variables.
data	An optional data frame containing the variables.
ID	A character string or expression specifying the group ID variable for multilevel models.
covariates	Optional numeric matrix or data frame of covariates to be included in the joint MVN model.
method	Correlation method for classic(): "pearson", "spearman", or "reml".
prior	Prior configuration object: ‘prior_flat()‘, ‘prior_normal()‘, or ‘prior_weak()‘. Default is ‘prior_flat()‘.
y_range	Optional numeric vector or matrix defining the theoretical range (min, max) of response variables. Required when using prior_weak(). Can be a vector of length 2 (applies to all variables) or a matrix/list of length P.
init	Optional list of initial values.
fixed	Optional named list of fixed values for specific parameters.
missing	Missing value handling strategy: "listwise" (default), "pairwise", or "fiml" (Full Information Maximum Likelihood).
WAIC	Logical; if TRUE, add pointwise ‘log_lik‘ to the generate block for WAIC.

Examples

```
# Estimate the correlation between two variables in the debate dataset
data(debate, package = "BayesRTMB")

fit_corr <- rtmb_corr(cbind(sat, perf), data = debate)


mcmc_corr <- fit_corr$sample(sampling = 500, warmup = 500, chains = 2)
mcmc_corr$summary()

bf_corr <- mcmc_corr$bayes_factor(fixed = list(corr = 0))
print(bf_corr)
```

---

rtmb_fa *RTMB-based Factor Analysis Wrapper*

Description

Performs exploratory factor analysis (EFA) using RTMB. It supports standard rotation methods (e.g., varimax, promax) as well as regularized factor analysis using a Spike-and-Slab Prior (SSP) for estimating sparse loading matrices.

Usage

```
rtmb_fa(
  data,
  nfactors = 1,
  rotate = NULL,
  score = FALSE,
  prior = prior_flat(),
  y_range = NULL,
  init = NULL,
  fixed = NULL,
  missing = c("listwise", "fiml"),
  WAIC = FALSE
)
```

Arguments

data	Observation data frame or matrix (N x J).
nfactors	Number of factors (K).
rotate	String specifying the rotation method (e.g., "varimax", "promax", "ssp"). If NULL, no rotation is applied. Specifying "ssp" performs regularized factor analysis.
score	Logical; if TRUE, factor scores are calculated in the generate block (default is FALSE).
prior	Prior configuration: ‘prior_flat()‘, ‘prior_normal()‘, or ‘prior_weak()‘. Hyperparameters can be specified within these functions (e.g., ‘prior_normal(mean_sd = 10, sd_rate = 10)‘). Available parameters for FA: ‘mean_sd‘, ‘sd_rate‘, ‘loadings_sd‘, and ‘ssp_ratio‘ (if ‘rotate = "ssp"‘).
y_range	A numeric vector of length 2 specifying the theoretical min and max values of the items. Used to construct weakly informative priors when ‘prior = prior_weak()‘.
init	List of initial values.
fixed	A named list of parameter values to fix (optional).
missing	Missing value handling strategy: "listwise" (default) or "fiml" (Full Information Maximum Likelihood).
WAIC	Logical; if TRUE, add pointwise ‘log_lik‘ to the generate block for WAIC.

Examples

```
# Prepare a subset of the BigFive dataset for factor analysis
data(BigFive, package = "BayesRTMB")
fa_data <- BigFive[, 1:10]

# --- 1. Standard Exploratory Factor Analysis (1 Factor) ---
fit_fa1 <- rtmb_fa(data = fa_data, nfactors = 1)

# Maximum A Posteriori (MAP) estimation
map_fa1 <- fit_fa1$optimize()
```

```
map_fa1$summary()



# --- 2. Factor Analysis with Rotation and Factor Scores (2 Factors) ---
# Extract 2 factors, apply Promax rotation during model fitting, and calculate factor scores
fit_fa2 <- rtmb_fa(data = fa_data, nfactors = 2, rotate = "promax", score = TRUE)

# Setting se_method = "sampling" enables standard errors and 95% CIs
# for transformed and generated quantities, such as factor scores and post-hoc rotations.
# (It uses multivariate normal sampling from the unconstrained parameter space)
map_fa2 <- fit_fa2$optimize(se_method = "sampling")
map_fa2$summary()

# Post-hoc rotation using the fa_rotate() method
# You can also apply other rotation methods (e.g., "varimax" from the stats package)
# to the unrotated loading matrix ("L") after estimation.
map_fa2$fa_rotate(target = "L", rotate = "varimax")

# The post-hoc rotated loadings are automatically stored with the method's
# suffix (e.g., "L_varimax")
map_fa2$summary("L_varimax")



# --- 3. Regularized Factor Analysis using Spike-and-Slab Prior (SSP) ---
# Specifying 'rotate = "ssp"' enables sparse loading matrix estimation
fit_ssp <- rtmb_fa(data = fa_data, nfactors = 2, rotate = "ssp")

map_ssp <- fit_ssp$optimize()
map_ssp$summary()
```

RTMB_Fit_Base *Base class for RTMB Fit objects*

Description

Base class for RTMB Fit objects

Base class for RTMB Fit objects

Details

An R6 base class providing common methods for Bayesian and MAP inference results.

Public fields

`model` The ‘RTMB_Model‘ object used for estimation.

Methods

Public methods:

- RTMB_Fit_Base$get_point_estimate()
- RTMB_Fit_Base$estimate()
- RTMB_Fit_Base$EAP()
- RTMB_Fit_Base$MAP()
- RTMB_Fit_Base$rotate()
- RTMB_Fit_Base$fa_rotate()
- RTMB_Fit_Base$clone()

Method `get_point_estimate()`**:** Abstract method to get a point estimate for a target parameter.

Usage:

```
RTMB_Fit_Base$get_point_estimate(target, ...)
```

Arguments:

`target` Character string specifying the target parameter name.

`...` Additional arguments.

Returns: A numeric array or matrix of the point estimate.

Method `estimate()`**:** Get point estimates for parameters, transformed parameters, and generated quantities.

Usage:

```
RTMB_Fit_Base$estimate(
  pars = NULL,
  type = c("mean", "EAP", "marginal_map", "joint_map", "MAP"),
  component = c("all", "parameters", "transform", "generate"),
  chains = NULL,
  best_chains = NULL,
  drop = TRUE,
  ...
)
```

Arguments:

`pars` Optional character or numeric vector of parameter names or indices to extract. Supports special keywords: "parameters", "transform", "generate", and "all".

`type` Character string specifying the estimation type.

`component` Character string specifying the component to filter by.

`chains` Numeric vector of chains to include.

`best_chains` Number of best chains to include.

`drop` Logical; if TRUE and only one parameter is selected, return the value directly instead of a list.

`...` Additional arguments passed to draws().

Returns: A named list of point estimates, or a single value if ‘drop = TRUE‘.

Method `EAP()`**:** Calculate Expected A Posteriori (EAP) estimates from posterior samples.

Usage:

```
RTMB_Fit_Base$EAP(
  pars = "parameters",
  chains = NULL,
  best_chains = NULL,
  drop = FALSE,
  ...
)
```

Arguments:

`pars` Optional character vector of parameter names to extract.

`chains` Numeric vector of chains to include.

`best_chains` Number of best chains to include.

`drop` Logical; whether to drop the list if only one parameter is selected.

`...` Additional arguments passed to 'estimate()'.

Returns: A named list of EAP estimates.

Method `MAP()`**:** Calculate Maximum A Posteriori (MAP) estimates.

Usage:

```
RTMB_Fit_Base$MAP(
  pars = "parameters",
  chains = NULL,
  best_chains = NULL,
  type = c("marginal", "joint"),
  drop = FALSE,
  ...
)
```

Arguments:

`pars` Optional character vector of parameter names to extract.

`chains` Numeric vector of chains to include.

`best_chains` Number of best chains to include.

`type` Character string; "marginal" or "joint" MAP.

`drop` Logical; whether to drop the list if only one parameter is selected.

`...` Additional arguments passed to 'estimate()'.

Returns: A named list of MAP estimates.

Method `rotate()`**:** Rotate sampled parameters.

Usage:

```
RTMB_Fit_Base$rotate(target, reference = NULL, linked = NULL)
```

Arguments:

`target` Character string specifying the target variable to base the rotation on.

`reference` Matrix to rotate towards. If NULL, the target's point estimate is used.

`linked` Character vector of variable names to be rotated in the same direction.

`overwrite` Logical; whether to overwrite the stored draws. If FALSE, adds to generated quantities. Default is FALSE.

`suffix` Character string to append to the rotated variable names when overwrite is FALSE. Default is "rot".

`...` Additional arguments passed to the rotation function.

Returns: The updated object invisibly.

Method `fa_rotate()`**:** Rotate factor loadings and optional factor scores.

Usage:

```
RTMB_Fit_Base$fa_rotate(
  target = "L",
  linked = NULL,
  scores = NULL,
  rotate = "promax",
  ...
)
```

Arguments:

`target` Character string specifying the target variable to base the rotation on.

`linked` Character vector of variable names to be rotated in the same direction.

`scores` Character vector of variable names to be rotated as factor scores (inverse direction).

`rotate` Character string specifying the rotation method.

`...` Additional arguments passed to the rotation function.

Returns: The updated object invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RTMB_Fit_Base$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`rtmb_glm` *RTMB-based GLM wrapper function (no random effects)*

Description

RTMB-based GLM wrapper function (no random effects)

Usage

```
rtmb_glm(
  formula,
  data,
  family = "gaussian",
  prior = prior_flat(),
  y_range = NULL,
  init = NULL,
```

```
  fixed = NULL,
  gmc = NULL,
  centering = NULL,
  view = NULL,
  factors = NULL,
  contrasts = "treatment",
  missing = c("listwise", "fiml"),
  WAIC = FALSE,
  ...
)
```

Arguments

formula	Formula
data	Data frame
family	Character string of the distribution family (e.g., "gaussian", "binomial", "poisson")
prior	An object of class '"rtmb_prior"'. Use 'prior_flat()' for no prior, 'prior_normal()' for default normal/exponential priors, or 'prior_weak()', 'prior_rhs()', 'prior_ssp()' for weakly informative or regularized Bayesian inference. Default is 'prior_flat()'.
y_range	Theoretical minimum and maximum values of the response variable as a vector c(min, max). Specifying this automatically enables weakly informative priors.
init	List of initial values
fixed	Optional named list of fixed values for specific parameters.
gmc	Character vector of variable names for GMC
centering	Alias for 'gmc'.
view	Optional character vector of parameter names to show first in summary.
factors	Character vector of variable names to be treated as factors.
contrasts	Character string specifying the contrast type ("treatment" or "sum").
missing	Missing value handling strategy: "listwise".
WAIC	Logical; if TRUE, add pointwise 'log_lik' to the generate block for WAIC.
...	Additional arguments passed to `rtmb_glmer()`.

Examples

```
# --- 1. Linear Regression (rtmb_lm) ---
# Fit a linear regression model using the debate dataset
data(debate, package = "BayesRTMB")
fit_lm <- rtmb_lm(sat ~ talk + perf, data = debate)
map_lm <- fit_lm$optimize()
map_lm$summary()

# --- 2. Generalized Linear Model (rtmb_glm) ---
# Fit a logistic regression model using the debate dataset
data(debate, package = "BayesRTMB")
```

```
fit_glm <- rtmb_glm(cond ~ talk + sat, data = debate, family = "bernoulli")
map_glm <- fit_glm$optimize()
map_glm$summary()

# --- 3. Generalized Linear Mixed Model (rtmb_glmer) ---
# Fit a linear mixed-effects model using the debate dataset
data(debate, package = "BayesRTMB")
fit_glmer <- rtmb_glmer(talk ~ cond + (1 | group), data = debate, family = "gaussian")

# MAP estimation using Laplace approximation for random effects
map_glmer <- fit_glmer$optimize(laplace = TRUE)
map_glmer$summary()

# MCMC sampling (chains and iterations reduced for faster execution)

mcmc_glmer <- fit_glmer$sample(sampling = 500, warmup = 500, chains = 2)
mcmc_glmer$summary()


# --- 4. Linear Mixed Model (rtmb_lmer) ---
# A convenient wrapper for Gaussian mixed models (identical to rtmb_glmer with family="gaussian")
fit_lmer <- rtmb_lmer(sat ~ talk + (1 | group), data = debate)
map_lmer <- fit_lmer$optimize()
map_lmer$summary()

# --- 5. Regularized Regression (Variable Selection) ---
# You can apply regularization to the fixed effects to shrink noise variables towards zero.
# Use prior = prior_rhs() for the Regularized Horseshoe prior,
# or prior_ssp() for the Spike-and-Slab prior.
# Note: When using regularization, you must specify 'y_range' (the theoretical minimum and maximum
# values of the response variable) to automatically set up the required weakly informative priors.

# Fit a linear regression using debate predictors with the Horseshoe prior
fit_rhs <- rtmb_lm(
  sat ~ talk + perf + skill,
  data = debate,
  prior = prior_rhs(),
  y_range = c(1, 5)
)
map_rhs <- fit_rhs$optimize()
# Summarize only the fixed effects (slopes)
map_rhs$summary("b")

# Fit a linear regression with the Spike-and-Slab prior
fit_ssp <- rtmb_lm(
  sat ~ talk + perf + skill,
  data = debate,
  prior = prior_ssp(),
  y_range = c(1, 5)
)
map_ssp <- fit_ssp$optimize()
map_ssp$summary("b")
```

`rtmb_glmer` *RTMB-based GLMM wrapper function*

Description

RTMB-based GLMM wrapper function

Usage

```
rtmb_glmer(
  formula,
  data,
  family = "gaussian",
  laplace = FALSE,
  prior = prior_flat(),
  y_range = NULL,
  init = NULL,
  fixed = NULL,
  gmc = NULL,
  centering = NULL,
  cwc = NULL,
  view = NULL,
  within = NULL,
  factors = NULL,
  contrasts = "treatment",
  sigma_by = NULL,
  resid_corr = NULL,
  resid_time = NULL,
  resid_group = NULL,
  generate = NULL,
  missing = c("listwise", "fiml"),
  WAIC = FALSE,
  .force_sum = FALSE
)
```

Arguments

`formula`	lme4-style formula (e.g., Y ~ X + (1 | GID))
`data`	Data frame
`family`	Character string of the distribution family (e.g., "gaussian", "binomial", "poisson", "ordered", "sequential")
`laplace`	Logical; whether to marginalize random effects using Laplace approximation
`prior`	An object of class "rtmb_prior" specifying the prior distribution. Use prior_weak(), prior_rhs(), or prior_ssp(). Default is ‘prior_flat()‘.

y_range	Theoretical minimum and maximum values of the response variable as a vector c(min, max). Required when using weakly informative or regularized priors with continuous models.
init	List of initial values (generated automatically based on glm if omitted)
fixed	Optional named list of fixed values for specific parameters.
gmc	Character vector of variable names for Grand Mean Centering (GMC). If "all", all numeric variables are centered.
centering	Alias for ‘gmc‘.
cwc	List for Centering Within Cluster (CWC). Should contain `cluster` (group variable) and `pars` (variable names to center).
view	Character vector of parameter names to prioritize in summary.
within	Optional list for wide-to-long conversion.
factors	Character vector of variable names to be treated as factors.
contrasts	Character string specifying the contrast type ("treatment" or "sum").
sigma_by	Character vector specifying variables to group residual variance by (heteroscedasticity).
resid_corr	Residual correlation structure: "ar1" (Autoregressive), "cs" (Compound Symmetry), "toep" (Toeplitz), or "un" (Unstructured).
resid_time	Variable name for time points in residual correlation.
resid_group	Variable name for grouping in residual correlation.
generate	Optional expression for generated quantities.
missing	Missing value handling strategy: "listwise".
WAIC	Logical; if TRUE, add pointwise ‘log_lik‘ to the generate block for WAIC.
.force_sum	Logical; internal use only.

Examples

```
# --- 1. Linear Regression (rtmb_lm) ---
# Fit a linear regression model using the debate dataset
data(debate, package = "BayesRTMB")
fit_lm <- rtmb_lm(sat ~ talk + perf, data = debate)
map_lm <- fit_lm$optimize()
map_lm$summary()

# --- 2. Generalized Linear Model (rtmb_glm) ---
# Fit a logistic regression model using the debate dataset
data(debate, package = "BayesRTMB")
fit_glm <- rtmb_glm(cond ~ talk + sat, data = debate, family = "bernoulli")
map_glm <- fit_glm$optimize()
map_glm$summary()

# --- 3. Generalized Linear Mixed Model (rtmb_glmer) ---
# Fit a linear mixed-effects model using the debate dataset
data(debate, package = "BayesRTMB")
fit_glmer <- rtmb_glmer(talk ~ cond + (1 | group), data = debate, family = "gaussian")
```

```
# MAP estimation using Laplace approximation for random effects
map_glmer <- fit_glmer$optimize(laplace = TRUE)
map_glmer$summary()

# MCMC sampling (chains and iterations reduced for faster execution)

mcmc_glmer <- fit_glmer$sample(sampling = 500, warmup = 500, chains = 2)
mcmc_glmer$summary()


# --- 4. Linear Mixed Model (rtmb_lmer) ---
# A convenient wrapper for Gaussian mixed models (identical to rtmb_glmer with family="gaussian")
fit_lmer <- rtmb_lmer(sat ~ talk + (1 | group), data = debate)
map_lmer <- fit_lmer$optimize()
map_lmer$summary()

# --- 5. Regularized Regression (Variable Selection) ---
# You can apply regularization to the fixed effects to shrink noise variables towards zero.
# Use prior = prior_rhs() for the Regularized Horseshoe prior,
# or prior_ssp() for the Spike-and-Slab prior.
# Note: When using regularization, you must specify 'y_range' (the theoretical minimum and maximum
# values of the response variable) to automatically set up the required weakly informative priors.

# Fit a linear regression using debate predictors with the Horseshoe prior
fit_rhs <- rtmb_lm(
  sat ~ talk + perf + skill,
  data = debate,
  prior = prior_rhs(),
  y_range = c(1, 5)
)
map_rhs <- fit_rhs$optimize()
# Summarize only the fixed effects (slopes)
map_rhs$summary("b")

# Fit a linear regression with the Spike-and-Slab prior
fit_ssp <- rtmb_lm(
  sat ~ talk + perf + skill,
  data = debate,
  prior = prior_ssp(),
  y_range = c(1, 5)
)
map_ssp <- fit_ssp$optimize()
map_ssp$summary("b")
```

---

rtmb_irt *RTMB-based IRT (Item Response Theory) Wrapper*

Description

Performs Item Response Theory modeling (1PL, 2PL, 3PL, and Graded Response Model) using RTMB. Missing values (NA) in the data are automatically removed internally for efficient computation.

Usage

```
rtmb_irt(
  data,
  model = c("2PL", "1PL", "3PL"),
  type = c("binary", "ordered"),
  prior = prior_flat(),
  init = NULL,
  fixed = NULL,
  view = NULL,
  missing = c("listwise", "fiml"),
  WAIC = FALSE
)
```

Arguments

data	A data frame or matrix of item responses (N persons x J items).
model	Character string for the model type: "1PL", "2PL", or "3PL".
type	Character string for the data type: "binary" or "ordered".
prior	Prior configuration: ‘prior_flat()‘, ‘prior_normal()‘, or ‘prior_weak()‘. Hyperparameters can be specified within these functions (e.g., ‘prior_normal(b_sd = 5)‘). Available parameters for IRT: ‘a_rate‘ (discrimination), ‘b_sd‘ (difficulty), ‘c_alpha‘/‘c_beta‘ (guessing).
init	List of initial values.
fixed	A named list of parameter values to fix (optional).
view	Character vector of parameter names to prioritize in summary.
missing	Missing value handling strategy: "listwise" (default) or "fiml" (Full Information Maximum Likelihood).
WAIC	Logical; if TRUE, add pointwise ‘log_lik‘ to the generate block for WAIC.

Examples

```
# --- 1. Binary Data (e.g., correct/incorrect answers) ---
# Simulate binary response data for 100 persons and 5 items
set.seed(123)
bin_data <- matrix(rbinom(500, size = 1, prob = 0.6), nrow = 100, ncol = 5)
colnames(bin_data) <- paste0("Item", 1:5)

# Introduce some missing values (NA) to demonstrate automatic handling
#bin_data[sample(1:500, 10)] <- NA

# Fit a 2-Parameter Logistic (2PL) model
```

```
fit_2pl <- rtmb_irt(data = bin_data, model = "2PL", type = "binary")

# Maximum A Posteriori (MAP) estimation
map_2pl <- fit_2pl$optimize()
map_2pl$summary()

# --- 2. Ordered Data (e.g., Likert scale) ---
# Simulate ordered response data (categories 1 to 5) with a true latent structure
set.seed(123)
N <- 100
J <- 5
K <- 4

# True parameters
theta <- rnorm(N, 0, 1)          # Person abilities (latent traits)
a <- runif(J, 0.8, 2.0)          # Item discriminations

# Item-specific thresholds (must be strictly increasing)
b <- matrix(NA, nrow = J, ncol = K - 1)
for (j in 1:J) {
  b[j, ] <- sort(c(-1.5, 0, 1.5) + rnorm(3, 0, 0.2))
}

ord_data <- matrix(NA, nrow = N, ncol = J)
colnames(ord_data) <- paste0("Item", 1:J)

# Generate responses based on the Graded Response Model
for (i in 1:N) {
  for (j in 1:J) {
    # Latent continuous response (eta + standard logistic noise)
    eta <- a[j] * theta[i]
    y_star <- eta + rlogis(1)

    # Apply thresholds to determine the observed category (1 to 4)
    category <- 1
    for (k in 1:(K - 1)) {
      if (y_star > b[j, k]) category <- k + 1
    }
    ord_data[i, j] <- category
  }
}

# Fit a Graded Response Model (2PL for ordered data)
fit_ord <- rtmb_irt(data = ord_data, model = "2PL", type = "ordered")
fit_ord$print_code()

map_ord <- fit_ord$optimize()
map_ord$summary()
```

---

rtmb_lm *RTMB-based Linear Regression wrapper function*

Description

RTMB-based Linear Regression wrapper function

Usage

```
rtmb_lm(
  formula,
  data,
  prior = prior_flat(),
  y_range = NULL,
  init = NULL,
  fixed = NULL,
  gmc = NULL,
  centering = NULL,
  view = NULL,
  factors = NULL,
  contrasts = "treatment",
  missing = c("listwise", "fiml"),
  WAIC = FALSE,
  ...
)
```

Arguments

formula	Formula (e.g., Y ~ X1 + X2)
data	Data frame
prior	An object of class '"rtmb_prior"'. Use 'prior_flat()' for no prior, 'prior_normal()' for default normal/exponential priors, or 'prior_weak()', 'prior_rhs()', 'prior_ssp()' for weakly informative or regularized Bayesian inference. Default is 'prior_flat()'.
y_range	Theoretical minimum and maximum values of the response variable as a vector c(min, max). Specifying this automatically enables weakly informative priors.
init	List of initial values.
fixed	Optional named list of fixed values for specific parameters.
gmc	Character vector of variable names for GMC
centering	Alias for 'gmc'.
view	Optional character vector of parameter names to show first in summary.
factors	Character vector of variable names to be treated as factors.
contrasts	Character string specifying the contrast type ("treatment" or "sum").
missing	Missing value handling strategy: "listwise".
WAIC	Logical; if TRUE, add pointwise 'log_lik' to the generate block for WAIC.
...	Additional arguments passed to `rtmb_glmer()`.

Examples

```
# --- 1. Linear Regression (rtmb_lm) ---
# Fit a linear regression model using the debate dataset
data(debate, package = "BayesRTMB")
fit_lm <- rtmb_lm(sat ~ talk + perf, data = debate)
map_lm <- fit_lm$optimize()
map_lm$summary()

# --- 2. Generalized Linear Model (rtmb_glm) ---
# Fit a logistic regression model using the debate dataset
data(debate, package = "BayesRTMB")
fit_glm <- rtmb_glm(cond ~ talk + sat, data = debate, family = "bernoulli")
map_glm <- fit_glm$optimize()
map_glm$summary()

# --- 3. Generalized Linear Mixed Model (rtmb_glmer) ---
# Fit a linear mixed-effects model using the debate dataset
data(debate, package = "BayesRTMB")
fit_glmer <- rtmb_glmer(talk ~ cond + (1 | group), data = debate, family = "gaussian")

# MAP estimation using Laplace approximation for random effects
map_glmer <- fit_glmer$optimize(laplace = TRUE)
map_glmer$summary()

# MCMC sampling (chains and iterations reduced for faster execution)

mcmc_glmer <- fit_glmer$sample(sampling = 500, warmup = 500, chains = 2)
mcmc_glmer$summary()


# --- 4. Linear Mixed Model (rtmb_lmer) ---
# A convenient wrapper for Gaussian mixed models (identical to rtmb_glmer with family="gaussian")
fit_lmer <- rtmb_lmer(sat ~ talk + (1 | group), data = debate)
map_lmer <- fit_lmer$optimize()
map_lmer$summary()

# --- 5. Regularized Regression (Variable Selection) ---
# You can apply regularization to the fixed effects to shrink noise variables towards zero.
# Use prior = prior_rhs() for the Regularized Horseshoe prior,
# or prior_ssp() for the Spike-and-Slab prior.
# Note: When using regularization, you must specify 'y_range' (the theoretical minimum and maximum
# values of the response variable) to automatically set up the required weakly informative priors.

# Fit a linear regression using debate predictors with the Horseshoe prior
fit_rhs <- rtmb_lm(
  sat ~ talk + perf + skill,
  data = debate,
  prior = prior_rhs(),
  y_range = c(1, 5)
)
map_rhs <- fit_rhs$optimize()
# Summarize only the fixed effects (slopes)
```

```
map_rhs$summary("b")

# Fit a linear regression with the Spike-and-Slab prior
fit_ssp <- rtmb_lm(
  sat ~ talk + perf + skill,
  data = debate,
  prior = prior_ssp(),
  y_range = c(1, 5)
)
map_ssp <- fit_ssp$optimize()
map_ssp$summary("b")
```

`rtmb_lmer` *RTMB-based Linear Mixed Model (LMM) wrapper function*

Description

RTMB-based Linear Mixed Model (LMM) wrapper function

Usage

```
rtmb_lmer(
  formula,
  data,
  laplace = TRUE,
  prior = prior_flat(),
  y_range = NULL,
  init = NULL,
  fixed = NULL,
  gmc = NULL,
  centering = NULL,
  cwc = NULL,
  view = NULL,
  sigma_by = NULL,
  factors = NULL,
  contrasts = "treatment",
  resid_corr = NULL,
  resid_time = NULL,
  resid_group = NULL,
  within = NULL,
  missing = c("listwise", "fiml"),
  WAIC = FALSE,
  ...
)
```

Arguments

formula	Formula
data	Data frame
laplace	Logical; whether to marginalize random effects using Laplace approximation
prior	An object of class '"rtmb_prior"'. Use 'prior_flat()' for no prior, 'prior_normal()' for default normal/exponential priors, or 'prior_weak()', 'prior_rhs()', 'prior_ssp()' for weakly informative or regularized Bayesian inference. Default is 'prior_flat()'.
y_range	Theoretical minimum and maximum values of the response variable
init	Initial values
fixed	Optional named list of fixed values for specific parameters.
gmc	Character vector of variable names for GMC
centering	Alias for 'gmc'.
cwc	List for CWC
view	Character vector of parameter names to prioritize in summary.
sigma_by	Character vector specifying variables to group residual variance by (heteroscedasticity).
factors	Character vector of variable names to be treated as factors.
contrasts	Character string specifying the contrast type ("treatment" or "sum").
resid_corr	Residual correlation structure (e.g., "ar1", "cs", "un", "toep").
resid_time	Variable name for time points in residual correlation.
resid_group	Variable name for grouping in residual correlation.
within	Optional list for wide-to-long conversion. For repeated measures data in wide format, specify the factor names and their levels, e.g., `list(Time = 4)` or `list(A = 2, B = 3)`. The total number of levels must match the number of columns in `cbind()` on the LHS. If omitted and the LHS is `cbind()`, the within-factor name is inferred from RHS variables not present in the data.
missing	Missing value handling strategy: "listwise".
WAIC	Logical; if TRUE, add pointwise 'log_lik' to the generate block for WAIC.
...	Additional arguments passed to `rtmb_model()`.

Value

RTMB_Model object

Examples

```
# --- 1. Linear Regression (rtmb_lm) ---
# Fit a linear regression model using the debate dataset
data(debate, package = "BayesRTMB")
fit_lm <- rtmb_lm(sat ~ talk + perf, data = debate)
map_lm <- fit_lm$optimize()
map_lm$summary()
```

```
# --- 2. Generalized Linear Model (rtmb_glm) ---
# Fit a logistic regression model using the debate dataset
data(debate, package = "BayesRTMB")
fit_glm <- rtmb_glm(cond ~ talk + sat, data = debate, family = "bernoulli")
map_glm <- fit_glm$optimize()
map_glm$summary()

# --- 3. Generalized Linear Mixed Model (rtmb_glmer) ---
# Fit a linear mixed-effects model using the debate dataset
data(debate, package = "BayesRTMB")
fit_glmer <- rtmb_glmer(talk ~ cond + (1 | group), data = debate, family = "gaussian")

# MAP estimation using Laplace approximation for random effects
map_glmer <- fit_glmer$optimize(laplace = TRUE)
map_glmer$summary()

# MCMC sampling (chains and iterations reduced for faster execution)

mcmc_glmer <- fit_glmer$sample(sampling = 500, warmup = 500, chains = 2)
mcmc_glmer$summary()


# --- 4. Linear Mixed Model (rtmb_lmer) ---
# A convenient wrapper for Gaussian mixed models (identical to rtmb_glmer with family="gaussian")
fit_lmer <- rtmb_lmer(sat ~ talk + (1 | group), data = debate)
map_lmer <- fit_lmer$optimize()
map_lmer$summary()

# --- 5. Regularized Regression (Variable Selection) ---
# You can apply regularization to the fixed effects to shrink noise variables towards zero.
# Use prior = prior_rhs() for the Regularized Horseshoe prior,
# or prior_ssp() for the Spike-and-Slab prior.
# Note: When using regularization, you must specify 'y_range' (the theoretical minimum and maximum
# values of the response variable) to automatically set up the required weakly informative priors.

# Fit a linear regression using debate predictors with the Horseshoe prior
fit_rhs <- rtmb_lm(
  sat ~ talk + perf + skill,
  data = debate,
  prior = prior_rhs(),
  y_range = c(1, 5)
)
map_rhs <- fit_rhs$optimize()
# Summarize only the fixed effects (slopes)
map_rhs$summary("b")

# Fit a linear regression with the Spike-and-Slab prior
fit_ssp <- rtmb_lm(
  sat ~ talk + perf + skill,
  data = debate,
  prior = prior_ssp(),
  y_range = c(1, 5)
)
```

```
map_ssp <- fit_ssp$optimize()
map_ssp$summary("b")
```

`rtmb_loglinear` *RTMB-based Log-linear analysis (Poisson regression)*

Description

Performs Bayesian or Frequentist log-linear analysis (Poisson regression) on a contingency table or raw data.

Usage

```
rtmb_loglinear(
  formula,
  data,
  prior = prior_flat(),
  fixed = NULL,
  WAIC = FALSE,
  ...
)
```

Arguments

`formula`	A formula (e.g., '~ A + B + A:B') or a contingency table.
`data`	A data frame (required if 'formula' is used).
`prior`	An object of class "rtmb_prior" specifying the prior distribution.
`fixed`	Optional named list of fixed values for specific parameters.
`WAIC`	Logical; if TRUE, add pointwise 'log_lik' to the generate block for WAIC.
`...`	Additional arguments passed to 'rtmb_glm()'.

Value

An 'RTMB_Model' object.

Examples

```
# Create a contingency table
tab <- matrix(c(10, 20, 30, 40), nrow = 2)
dimnames(tab) <- list(A = c("A1", "A2"), B = c("B1", "B2"))

# Fit a log-linear model (independence model: ~ A + B)
fit_log <- rtmb_loglinear(~ A + B, data = tab)

# MAP estimation
map_log <- fit_log$optimize()
map_log$summary()
```

rtmb_lrt *Fit a Latent Rank Theory (LRT) Model*

Description

Fits a Latent Rank Theory model, which is a mixture model with ordered ranks and Gaussian Process smoothing on the mean profiles.

Usage

```
rtmb_lrt(
  formula,
  k = 3,
  data = NULL,
  rank_coords = NULL,
 covariance = c("diagonal", "diagonal_equal", "full", "full_equal", "full_equal_corr"),
  magnitude = NULL,
  smoothing = NULL,
  noise = 0.01,
  prob_smoothing = FALSE,
  link = c("ordered", "sequential"),
  prior = prior_flat(),
  y_range = NULL,
  fixed = NULL,
  two_stage = FALSE,
  WAIC = FALSE,
  ...
)
```

Arguments

formula	A formula specifying the response variable(s).
k	Number of ranks (mixture components).
data	A data frame containing the variables.
rank_coords	Optional numeric vector of coordinates for each rank. Default is 1:k.
covariance	Covariance structure: "diagonal", "diagonal_equal", "full", "full_equal", or "full_equal_corr".
magnitude	Signal standard deviation for the GP prior. If NULL, it is estimated.
smoothing	Length-scale for the GP prior. If NULL, it is estimated.
noise	Measurement noise for the GP prior (default is 0.01).
prob_smoothing	Logical; whether to apply smoothing to the class membership probabilities.
link	Link function for class probabilities: "ordered" or "sequential".
prior	Prior configuration: ‘prior_flat()‘, ‘prior_normal()‘, ‘prior_weak()‘, ‘prior_rhs()‘, or ‘prior_ssp()‘. Default is ‘prior_flat()‘. If ‘y_range‘ is supplied with the default flat prior, the wrapper automatically switches to ‘prior_weak()‘.

y_range	Optional numeric vector or matrix defining the theoretical range (min, max) of response variables. Specifying this automatically enables weakly informative priors if ‘prior‘ is ‘prior_flat()‘.
fixed	Optional named list of fixed values for specific parameters.
two_stage	Logical; if TRUE, estimate the latent-rank measurement model first and then estimate the rank regression with delta-method uncertainty propagation. Currently supported for ‘$optimize()‘ only.
WAIC	Logical; if TRUE, add pointwise ‘log_lik‘ to the generate block for WAIC.
...	Additional arguments passed to ‘rtmb_model‘.

Value

A RTMB_Model object.

---

rtmb_mdu *RTMB-based Multidimensional Unfolding Wrapper*

Description

Fits a multidimensional unfolding model for preference/rating data. Rows are persons or observations and columns are stimuli/items. The model represents both row scores (‘theta‘) and item locations (‘delta‘) in a shared D-dimensional space.

Usage

```
rtmb_mdu(
  data,
  ndim = 2,
  distance = c("squared", "euclidean"),
  alpha = c("random", "fix"),
  method = c("rating", "Best", "Best-Worst", "MDS"),
  sets = NULL,
  prior = prior_flat(),
  y_range = NULL,
  init = NULL,
  fixed = NULL,
  view = NULL,
  distance_eps = 1e-04,
  missing = c("listwise", "fiml"),
  WAIC = FALSE
)
```

Arguments

data	Numeric matrix or data frame (N rows x M items) for ‘method = "rating"‘. For choice methods, a list containing ‘Best‘ and, for ‘method = "Best-Worst"‘, ‘Worst‘. For ‘method = "Best-Worst"‘, a single matrix/data frame is treated as pre-coded ‘Y_dif‘ pair indices.
ndim	Number of unfolding dimensions.
distance	Character; ‘"squared"‘ uses squared Euclidean distance (default, often easier for optimization), while ‘"euclidean"‘ uses Euclidean distance.
alpha	Character; ‘"random"‘ estimates item-specific alpha values as random effects (default), while ‘"fix"‘ estimates a single common alpha.
method	Character; ‘"rating"‘ for continuous ratings, ‘"Best"‘ for best-only choice tasks, ‘"Best-Worst"‘ for best-worst choice tasks, or ‘"MDS"‘ for fitting a multidimensional scaling model to a distance matrix.
sets	Matrix or data frame of presented item sets (P tasks x C items) for choice methods.
prior	Prior configuration: ‘prior_flat()‘, ‘prior_normal()‘, or ‘prior_weak()‘. ‘prior_flat()‘ creates a maximum-likelihood model suitable for ‘classic()‘. The latent coordinates ‘delta‘ and ‘theta‘ are always treated as random effects with normal scale priors, similarly to IRT ability parameters.
y_range	Optional response range for ‘method = "rating"‘. If supplied with the default flat prior, ‘prior_weak()‘ is used. For choice methods (‘"Best"‘ and ‘"Best-Worst"‘), ‘prior_weak()‘ behaves like ‘prior_normal()‘ because there is no observed rating scale.
init	Optional named list of initial values.
fixed	Optional named list of parameter values to fix.
view	Character vector of parameter names to prioritize in summaries.
distance_eps	Small positive constant added to the distance.
missing	Missing value handling strategy: "listwise" (default) or "fiml" (Full Information Maximum Likelihood).
WAIC	Logical; if TRUE, add pointwise ‘log_lik‘ to the generate block for WAIC.

Value

An ‘RTMB_Model‘ object.

Examples

```
# Simulate rating data for Multidimensional Unfolding (MDU)
set.seed(123)
N <- 50  # Number of persons
M <- 10  # Number of items
D <- 2   # Number of dimensions

# True person and item coordinates in a 2D space
theta <- matrix(rnorm(N * D), N, D)
```

```
delta <- matrix(rnorm(M * D), M, D)

# Generate distance-like ratings (smaller means more preferred)
Y <- matrix(NA, N, M)
for(i in 1:N) {
  for(j in 1:M) {
    Y[i, j] <- sum((theta[i,] - delta[j,])^2) + rnorm(1, 0, 0.5)
  }
}

# Fit a 2-dimensional MDU model
fit_mdu <- rtmb_mdu(Y, ndim = 2, distance = "squared", method = "rating")

# MAP estimation
map_mdu <- fit_mdu$optimize()
map_mdu$summary()

# Note: MDU models have many parameters, so MCMC sampling might take time.

# mcmc_mdu <- fit_mdu$sample(sampling = 500, warmup = 500, chains = 2)
```

`rtmb_mediation` *RTMB-based Mediation Analysis Wrapper*

Description

'rtmb_mediation' performs mediation analysis by simultaneously estimating multiple GLM regression equations. It automatically identifies mediation paths and calculates indirect, direct, and total effects.

Usage

```
rtmb_mediation(
  formula,
  data,
  family = "gaussian",
  prior = prior_flat(),
  y_range = NULL,
  fixed = NULL,
  view = NULL,
  WAIC = FALSE,
  ...
)
```

Arguments

`formula`	A list of formulas defining the regression paths (e.g., 'list(M ~ X, Y ~ X + M)').
`data`	A data frame containing the variables.

family	A single character string or a list of character strings specifying the error distribution for each equation (e.g., ‘family = list("gaussian", "binomial")‘). Default is "gaussian".
prior	An object of class "rtmb_prior" specifying the prior distribution.
y_range	Theoretical minimum and maximum values of the response variable.
fixed	A named list of parameter values to fix (optional).
view	Character vector of parameter names to prioritize in summary.
WAIC	Logical; if TRUE, add pointwise ‘log_lik‘ to the generate block for WAIC.
...	Additional arguments passed to the model construction.

Details

The function identifies mediation paths by looking for variables that are responses in one equation and predictors in another. Indirect effects are calculated as the product of coefficients along these paths ($a * b$).

Uncertainty Estimation: When using ‘$optimize(ci_method = "sampling")‘, the function provides asymmetric confidence intervals for indirect effects based on the distribution of products, which is more accurate than the standard Sobel test (Delta Method).

Value

An ‘RTMB_Model‘ object.

Examples

```
# Simulate mediation data
set.seed(123)
N <- 100
X <- rnorm(N)
# M is influenced by X
M <- 0.5 * X + rnorm(N, 0, 0.5)
# Y is influenced by both X and M
Y <- 0.3 * X + 0.8 * M + rnorm(N, 0, 0.5)
dat <- data.frame(X, M, Y)

# Fit a mediation model
# The formula list specifies the two regression equations
fit_med <- rtmb_mediation(list(M ~ X, Y ~ X + M), data = dat)

# Maximum A Posteriori (MAP) estimation
map_med <- fit_med$optimize()
# The summary automatically calculates indirect, direct, and total effects
map_med$summary()
```

`rtmb_mixture` *Mixture Model Wrapper for RTMB*

Description

Provides a user-friendly interface for fitting Gaussian mixture models with optional covariates on class membership probabilities and various covariance structures.

Usage

```
rtmb_mixture(
  formula,
  k = 2,
  data = NULL,
 covariance = c("diagonal", "diagonal_equal", "full", "full_equal", "full_equal_corr"),
  prior = prior_flat(),
  y_range = NULL,
  fixed = NULL,
  WAIC = FALSE,
  ...
)
```

Arguments

`formula`	A formula specifying the response variable(s). For multivariate, use `cbind(y1, y2) ~ 1`.
`k`	Number of mixture components.
`data`	A data frame containing the variables in the model.
`covariance`	Covariance structure: "diagonal" (default), "diagonal_equal", "full", "full_equal", or "full_equal_corr".
`prior`	Prior configuration: ‘prior_flat()‘, ‘prior_normal()‘, ‘prior_weak()‘, ‘prior_rhs()‘, or ‘prior_ssp()‘. Default is ‘prior_flat()‘. If ‘y_range‘ is supplied with the default flat prior, the wrapper automatically switches to ‘prior_weak()‘.
`y_range`	Optional numeric vector or matrix defining the theoretical range (min, max) of response variables. Specifying this automatically enables weakly informative priors if ‘prior‘ is ‘prior_flat()‘.
`fixed`	Optional named list of fixed values for specific parameters.
`WAIC`	Logical; if TRUE, add pointwise ‘log_lik‘ to the generate block for WAIC.
`...`	Additional arguments passed to ‘rtmb_model‘.

Value

A `RTMB_Model` object.

Examples

```
# Simulate 1D mixture data (2 components)
set.seed(123)
N <- 100
group <- rbinom(N, 1, 0.4)
y <- ifelse(group == 1, rnorm(N, 5, 1), rnorm(N, 0, 1))
dat <- data.frame(y)

# Fit a 1D Gaussian mixture model with 2 components
fit_mix <- rtmb_mixture(y ~ 1, k = 2, data = dat)

# MAP estimation
map_mix <- fit_mix$optimize()
map_mix$summary()
```

---

rtmb_model *Create an RTMB_Model Object*

Description

The ‘rtmb_model‘ function acts as the core constructor for compiling and combining user-defined data with the model code (defined via ‘rtmb_code‘). It generates an ‘RTMB_Model‘ (R6 class) instance, which serves as the foundation for performing Bayesian inference, including Maximum A Posteriori (MAP) estimation, Variational Inference (ADVI), and Markov Chain Monte Carlo (MCMC) sampling.

Usage

```
rtmb_model(
  data,
  code,
  par_names = list(),
  init = NULL,
  view = NULL,
  fixed = NULL,
  silent = FALSE
)
```

Arguments

`data`	A named list containing observation data and constants (e.g., sample size, matrices) used in the model.
`code`	A model definition block generated by `rtmb_code(...)` (must include `parameters` and `model` blocks).
`par_names`	A named list of character vectors corresponding to the dimensions of specific parameters (optional).

`init`	A list or numeric vector of initial values for parameters (optional). If not specified, initialized randomly.
`view`	Character vector of parameter names to be displayed preferentially at the top when outputting results like `summary()` (optional).
`fixed`	A named list of parameter values to fix (optional). Useful for scoring or plug-in estimation where some parameters (e.g., item parameters) are fixed to known values.
`silent`	Logical; if TRUE, suppresses diagnostic messages during model creation. Default is FALSE.

Details

Model Compilation and Pre-checking: When this function is called, it evaluates the provided data and model blocks. It performs a "sandbox execution" (pre-check) using dummy initial values to dynamically detect common structural errors—such as undefined variables, out-of-bounds indices, or incompatible matrix operations—before proceeding to the computationally expensive Automatic Differentiation (MakeADFun) phase. Cryptic backend errors are caught and translated into user-friendly hints.

Writing AD-Compatible Code (Important): To ensure the model is differentiable, you must follow specific syntax rules when writing code within `rtmb_code`. Avoid discrete branching (`if`, `ifelse`) based on parameters and use numerically stable functions. See `rtmb_syntax` for a detailed guide on Automatic Differentiation requirements.

Initial Values (`init`): Initial parameter values can be specified as a flat numeric vector or a named list. If a partial list is provided, the unspecified parameters are automatically initialized with random values drawn from a uniform distribution on the unconstrained scale. If `init = NULL`, all parameters are initialized randomly.

Parameter Labeling (`par_names`): By default, vector or matrix parameters are displayed with numeric indices (e.g., `mu[1]`). You can pass a named list of character vectors to `par_names` to assign meaningful labels to specific dimensions (e.g., `mu[Control]`), vastly improving the readability of summary outputs and trace plots.

Available Inference Methods on the Returned Object: The returned `RTMB_Model` instance provides the following core methods:

- `$optimize(...)`: Performs Maximum A Posteriori (MAP) or Maximum Likelihood estimation. Returns a `MAP_Fit` object.
- `$sample(...)`: Draws posterior samples using the NUTS (No-U-Turn Sampler) algorithm. Returns an `MCMC_Fit` object.
- `$variational(...)`: Performs Mean-field or Full-rank Automatic Differentiation Variational Inference (ADVI). Returns a `VB_Fit` object.

Value

An `RTMB_Model` class instance with a compiled and pre-tested automatic differentiation function.

An `RTMB_Model` class instance with a compiled and pre-tested automatic differentiation function.

Examples

```
# Simulate data for 3 groups
set.seed(123)
N <- 60
group_idx <- sample(1:3, N, replace = TRUE)
group_names <- c("Control", "Treatment_A", "Treatment_B")

# True group means: Control = 0, Treatment_A = 2, Treatment_B = -1
true_means <- c(0, 2, -1)
y <- true_means[group_idx] + rnorm(N, mean = 0, sd = 0.5)

data_list <- list(N = N, K = 3, group_idx = group_idx, y = y)

# Define the model using rtmb_code
model_code <- rtmb_code(
  parameters = {
    mu    = Dim(K)             # Vector of length K (group means)
    sigma = Dim(1, lower = 0)  # Scalar (residual standard deviation)
  },
  model = {
    # Priors
    for (k in 1:K) mu[k] ~ normal(0, 10)
    sigma ~ exponential(1)

    # Likelihood
    for (i in 1:N) {
      y[i] ~ normal(mu[group_idx[i]], sigma)
    }
  }
)

# --- 1. Basic Model Creation ---
# Create the RTMB_Model object
mod_basic <- rtmb_model(
  data = data_list,
  code = model_code
)

# Perform Maximum A Posteriori (MAP) estimation
map_basic <- mod_basic$optimize()
# The summary displays default parameter names: mu[1], mu[2], mu[3]
map_basic$summary()

# --- 2. Optional: Adding Custom Parameter Names and initial values ---
# You can optionally use 'par_names' to assign meaningful labels
# to vector or matrix elements for easier interpretation.
mod_named <- rtmb_model(
  data = data_list,
  code = model_code,
  init = list(mu = rep(0, 3), sigma = 1),
  par_names = list(mu = group_names)
)
```

```
map_named <- mod_named$optimize()
# The summary now displays: mu[Control], mu[Treatment_A], mu[Treatment_B]
map_named$summary()
```

`RTMB_Model-class` *RTMB model object*

Description

An R6 class representing a Bayesian model built with RTMB. This class stores model components and provides methods for building the automatic differentiation object, optimizing the posterior, and drawing posterior samples.

Public fields

`data` Named list containing observation data.

`par_list` Evaluated list of parameter definitions.

`log_prob` Executable function for likelihood and prior calculations.

`transform` Function for calculating transformed parameters.

`generate` Function for calculating generated quantities.

`par_names` List of variable names corresponding to parameter dimensions.

`pl_full` Complete parameter list with expanded constraints and dimension information used internally.

`formula` Formula used for model generation (if created via a wrapper function).

`requested_contrasts` Character; user-requested contrast setting (e.g. "treatment").

`contrasts` Character; internal contrast setting used for fitting (e.g. "sum").

`raw_data` Data frame; the original data used for the model.

`family` Character string of the distribution family (if created via a wrapper function).

`init` List or numeric vector of initial values.

`view` Character vector of variable names to prioritize in summary.

`code` Original model AST generated by 'rtmb_code()'.

`map` A list specifying parameter mappings for fixing values (e.g., via `fixed_model()`).

`prior_correction` Numeric value for legacy or low-level marginal-likelihood correction. Fixed-parameter prior removal is handled during AD objective construction.

`fixed_prior_specs` Internal specification for removing fixed-parameter prior contributions during AD evaluation.

`type` Character; the type of the model (e.g. 'lmer', 'glm', 'table', 'ttest', 'corr', 'mediation').

`extra` List; used to store auxiliary data (like term assignments, factors, test results) needed by methods such as classic().

Methods

Public methods:

- RTMB_Model$new()
- RTMB_Model$prepare_init()
- RTMB_Model$get_par_list()
- RTMB_Model$get_ad_obj()
- RTMB_Model$to_unconstrained()
- RTMB_Model$to_constrained()
- RTMB_Model$unconstrained_vector_to_list()
- RTMB_Model$constrained_vector_to_list()
- RTMB_Model$calculate_satterthwaite_df()
- RTMB_Model$calculate_reml_satterthwaite_df()
- RTMB_Model$build_ad_obj()
- RTMB_Model$optimize()
- RTMB_Model$classic()
- RTMB_Model$sample()
- RTMB_Model$variational()
- RTMB_Model$print_code()
- RTMB_Model$print_log_prob()
- RTMB_Model$print_transform()
- RTMB_Model$print_generate()
- RTMB_Model$get_n_obs()
- RTMB_Model$fixed_model()
- RTMB_Model$clone()

Method `new()`**:** Create a new 'RTMB_Model' object.

Usage:

```
RTMB_Model$new(
  data,
  par_list,
  log_prob,
  transform = NULL,
  generate = NULL,
  par_names = NULL,
  init = NULL,
  view = NULL,
  code = NULL
)
```

Arguments:

`data` Named list containing observation data.

`par_list` Evaluated list of parameter definitions.

`log_prob` Executable function for likelihood and prior calculations.

`transform` Function for calculating transformed parameters (optional).

`generate` Function for calculating generated quantities (optional).

`par_names` List of variable names corresponding to parameter dimensions.

`init` List or numeric vector of initial values.

`view` Character vector of variable names to prioritize in summary.

`code` Original model AST generated by 'rtmb_code()'.

Method `prepare_init()`**:** Prepare and format initial values for the model parameters.

Usage:

```
RTMB_Model$prepare_init(init_arg)
```

Arguments:

`init_arg` Optional list or numeric vector of initial values. If NULL, defaults to 'self$init' or random generation.

Returns: A flat numeric vector of constrained initial values.

Method `get_par_list()`**:** Get the current parameters as a named list.

Usage:

```
RTMB_Model$get_par_list(init = NULL)
```

Arguments:

`init` Optional numeric vector or list of initial values. If NULL, uses the model's current initial values.

Returns: A named list of constrained parameters.

Method `get_ad_obj()`**:** Get the RTMB automatic differentiation object.

Usage:

```
RTMB_Model$get_ad_obj(...)
```

Arguments:

`...` Arguments passed to `build_ad_obj`.

Returns: A TMB objective object (ad_obj).

Method `to_unconstrained()`**:** Convert a list of constrained parameters to unconstrained space.

Usage:

```
RTMB_Model$to_unconstrained(par_list_constrained)
```

Arguments:

`par_list_constrained` A named list of parameters in their natural space.

Returns: A named list of parameters in unconstrained space.

Method `to_constrained()`**:** Convert a list of unconstrained parameters to constrained (natural) space.

Usage:

```
RTMB_Model$to_constrained(par_list_unconstrained)
```

Arguments:

`par_list_unconstrained` A named list of parameters in unconstrained space.

Returns: A named list of parameters in constrained space.

Method `unconstrained_vector_to_list()`**:** Convert a flat unconstrained vector to a named list.

Usage:

```
RTMB_Model$unconstrained_vector_to_list(vec)
```

Arguments:

`vec` A numeric vector in unconstrained space.

Returns: A named list of unconstrained parameters.

Method `constrained_vector_to_list()`**:** Convert a flat constrained vector to a named list.

Usage:

```
RTMB_Model$constrained_vector_to_list(vec)
```

Arguments:

`vec` A numeric vector in constrained space.

Returns: A named list of constrained parameters.

Method `calculate_satterthwaite_df()`**:** Calculate Satterthwaite degrees of freedom for parameters.

Usage:

```
RTMB_Model$calculate_satterthwaite_df(
  ad_obj,
  idx_fix_active = NULL,
  L_u_total = NULL,
  opt_par = NULL,
  max_df = NULL,
  silent = FALSE,
  return_sensitivities = FALSE
)
```

Arguments:

`ad_obj` An RTMB objective object at the optimum.

`idx_fix_active` Integer vector; indices of active fixed parameters in the full unconstrained vector.

`L_u_total` Integer; total length of the full unconstrained parameter vector.

`opt_par` Optional numeric vector of optimized parameters.

`max_df` Numeric; maximum allowed degrees of freedom. Default is NULL.

`silent` Logical; whether to suppress informational messages. Default is FALSE.

`return_sensitivities` Logical; if TRUE, returns Hessian sensitivities and V for delta method.

Returns: A numeric vector of estimated degrees of freedom (length = L_u_total). Inf for random effects.

Method `calculate_reml_satterthwaite_df()`**:** Calculate Satterthwaite degrees of freedom for integrated fixed effects (REML).

Usage:

```
RTMB_Model$calculate_reml_satterthwaite_df(
  ad_obj,
  opt_par,
  beta_idx,
  max_df = NULL,
  silent = FALSE,
  return_sensitivities = FALSE
)
```

Arguments:

`ad_obj` An RTMB objective object.

`opt_par` Numeric vector of optimized variance components.

`beta_idx` Integer vector; indices of fixed effects within the random effects vector.

`max_df` Numeric; maximum allowed degrees of freedom. Default is NULL.

`silent` Logical; whether to suppress informational messages. Default is FALSE.

`return_sensitivities` Logical; if TRUE, returns Hessian sensitivities and delta-method components.

`...` Additional arguments.

Returns: A numeric vector of estimated degrees of freedom for the fixed effects.

Method `build_ad_obj()`**:** Build the RTMB automatic differentiation object.

Usage:

```
RTMB_Model$build_ad_obj(
  init = NULL,
  laplace = FALSE,
  jacobian_target = "none",
  map = NULL,
  .marginal_vars = NULL
)
```

Arguments:

`init` Optional numeric vector or list of initial values for the parameters. Default is NULL.

`laplace` Logical; whether to use Laplace approximation to marginalize random effects. Default is FALSE.

`jacobian_target` Character string specifying which parameters to apply Jacobian adjustments to:

- `"all"`: Apply to all parameters (standard for Bayesian MCMC).
- `"random"`: Apply only to parameters with `random = TRUE` (standard for Laplace approximation/REML).
- `"none"`: No Jacobian adjustment (standard for Maximum Likelihood estimation).

Default is "none".

`map` Optional list specifying parameters to fix. Passed directly to MakeADFun. Default is NULL.

`.marginal_vars` Internal use for profiling marginalized parameters.

Returns: An RTMB objective object (ad_obj).

Method `optimize()`**:** Perform Maximum Likelihood or Maximum A Posteriori (MAP) estimation.

Usage:

```
RTMB_Model$optimize(
  laplace = TRUE,
  init = NULL,
  num_estimate = 1,
  control = list(),
  optimizer = "nlminb",
  method = "BFGS",
  map = NULL,
  fixed = NULL,
  se_method = c("wald", "sampling", "none"),
  num_samples = 1000,
  seed = 123,
  marginal = "none",
  df_method = "auto",
  view = NULL,
  ...
)
```

Arguments:

`laplace` Logical; whether to use Laplace approximation for random effects. Default is TRUE.

`init` Optional initial values for parameters (numeric vector or list).

`num_estimate` Integer; Number of multi-start optimization runs. Default is 1.

`control` A list of control settings passed to the optimizer.

`optimizer` Character; The optimizer to use: "nlminb" (default) or "optim".

`method` Character; The method for "optim" (e.g., "BFGS", "L-BFGS-B").

`map` Optional list specifying parameters to fix (factors).

`fixed` Optional list specifying parameter values to fix.

`se_method` Character; method for uncertainty estimation.

- `"wald"`: Wald standard errors and Wald confidence intervals.
- `"sampling"`: Simulation-based error propagation.
- `"none"`: Do not compute standard errors, confidence intervals, or degrees of freedom.

`num_samples` Integer; number of samples to draw when se_method is "sampling". Default is 1000.

`seed` Integer; random seed for sampling.

`marginal` Character vector or "auto" or "none"; specifies which parameters to marginalize out via Laplace approximation.

- `"none"` (default): No additional marginalization.
- `"auto"`: Uses variables specified by the model wrapper (e.g., fixed effects in lmer).
- `"all"`: Marginalize all parameters (advanced use).
- `character vector`: Specific parameter names to marginalize.

`df_method` Character or numeric; method for finite degrees-of-freedom approximation. Used only when ‘marginal‘ resolves to one or more parameters and ‘se_method != "none"‘.

- "auto": Uses Satterthwaite approximation when marginalization is active.
- "satterthwaite": Satterthwaite approximation.
- "inf" or "none": Use infinite degrees of freedom.
- numeric: Use the supplied fixed degrees of freedom.

view Character vector of parameter names to prioritize in summary display.

... Additional arguments.

Returns: A fitted 'MAP_Fit' object.

Method classic(): Perform frequentist inference (REML/ML) with model-appropriate degrees of freedom.

Usage:

```
RTMB_Model$classic(
  df_method = "auto",
  se_method = c("wald", "sampling"),
  num_samples = 1000,
  seed = 123,
  view = NULL,
  map = NULL,
  fixed = NULL,
  ...
)
```

Arguments:

df_method Character; Method for calculating degrees of freedom. Default is "auto".

- "auto": Determined based on model type (e.g., Satterthwaite for mixed models).
- "residual": Residual degrees of freedom.
- "satterthwaite": Satterthwaite approximation (robust for mixed models).
- "inf": Use z-distribution (infinite degrees of freedom).

se_method Character; The method for CI estimation: "wald" (default) or "sampling".

num_samples Integer; number of samples to draw when se_method is "sampling". Default is 1000.

seed Integer; random seed for sampling.

view Character vector of parameters to prioritize in the summary output.

map Optional list specifying parameters to fix.

fixed Optional list specifying parameter values to fix.

... Additional arguments.

Returns: A 'Classic_Fit' object.

Method sample(): Draw posterior samples from the model.

Usage:

```
RTMB_Model$sample(
  sampling = 1000,
  warmup = 1000,
  chains = 4,
  thin = 1,
```

```
  seed = sample.int(1e+06, 1),
  delta = 0.8,
  max_treedepth = 10,
  parallel = FALSE,
  laplace = FALSE,
  init = NULL,
  init_jitter = 0.1,
  save_csv = NULL,
  map = NULL,
  fixed = NULL
)
```

Arguments:

`sampling` Number of sampling iterations. Default is 1000.

`warmup` Number of warmup iterations. Default is 1000.

`chains` Number of MCMC chains. Default is 4.

`thin` Thinning interval. Default is 1.

`seed` Random seed.

`delta` Target acceptance rate for HMC/NUTS. Default is 0.8.

`max_treedepth` Maximum tree depth for HMC/NUTS. Default is 10.

`parallel` Logical; whether to run chains in parallel. Default is FALSE.

`laplace` Logical; whether to use Laplace approximation. Default is FALSE.

`init` Optional initial values for parameters.

`init_jitter` sd of randomize initial values for parameters.

`save_csv` Optional list for saving MCMC results. e.g., list(name = "model", dir = "BayesRTMB_mcmc").

`map` Optional list specifying parameters to fix (factors).

`fixed` Optional list specifying parameter values to fix.

Returns: A fitted 'MCMC_Fit' object.

Method `variational()`**:** Run Automatic Differentiation Variational Inference (ADVI).

Usage:

```
RTMB_Model$variational(
  iter = 3000,
  tol_rel_obj = 0.005,
  window_size = 100,
  num_samples = 1000,
  num_estimate = 4,
  alpha = 0.01,
  laplace = FALSE,
  print_freq = 1000,
  method = c("meanfield", "fullrank", "hybrid"),
  parallel = FALSE,
  seed = sample.int(1e+06, 1),
  init = NULL,
  save_csv = NULL,
  map = NULL,
  fixed = NULL
)
```

Arguments:

`iter` Integer; fixed number of iterations for the optimization. Default is 3000.

`tol_rel_obj` Numeric; relative tolerance for the ELBO change to determine convergence. Default is 0.005.

`window_size` Integer; window size for median smoothing in the convergence check. Default is 100.

`num_samples` Integer; number of posterior samples to generate from the fitted variational distribution. Default is 1000.

`num_estimate` Integer; number of times to run the VB estimation (treated as chains). Default is 4.

`alpha` Numeric; learning rate for the Adam optimizer. Default is 0.01.

`laplace` Logical; whether to use Laplace approximation to marginalize random effects. Default is FALSE.

`print_freq` Integer; iterations interval for progress output. Set to 0 to disable. Default is 1000.

`method` Character; method of Variational Inference. Default is "meanfield".

`parallel` Logical; whether to run estimations in parallel. Default is FALSE.

`seed` Integer; random seed for reproducibility.

`init` Optional numeric vector or list for initial parameter values. Default is NULL.

`save_csv` Optional list for saving VB results. e.g., list(name = "model", dir = "BayesRTMB_vb").

`map` Optional list specifying parameters to fix (factors).

`fixed` Optional list specifying parameter values to fix.

Returns: A fitted ‘VB_Fit‘ object containing posterior samples and diagnostic information.

Method `print_code()`**:** Print model code or model structure.

Usage:

```
RTMB_Model$print_code()
```

Returns: The object itself, invisibly.

Method `print_log_prob()`**:** Print the internal log-probability function.

Usage:

```
RTMB_Model$print_log_prob()
```

Returns: The object itself, invisibly.

Method `print_transform()`**:** Print the internal transformation function.

Usage:

```
RTMB_Model$print_transform()
```

Returns: The object itself, invisibly.

Method `print_generate()`**:** Print the internal generation function.

Usage:

```
RTMB_Model$print_generate()
```

Returns: The object itself, invisibly.

Method `get_n_obs()`**:** Automatically detect and count the number of independent data points (observations).

Usage:

```
RTMB_Model$get_n_obs()
```

Returns: Integer; total number of observations, or NULL if model code is missing.

Method `fixed_model()`**:** Create a model with fixed parameters.

Usage:

```
RTMB_Model$fixed_model(fixed = list(), silent = FALSE)
```

Arguments:

`fixed` A named list of parameter values to fix.

`silent` Logical; whether to suppress informational messages. Default is FALSE.

Returns: A new RTMB_Model object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RTMB_Model$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`rtmb_syntax` *Guidelines for Writing RTMB-Compatible Code*

Description

Models defined in `rtmb_code` rely on Automatic Differentiation (AD) via the `RTMB` package. To ensure the model is differentiable and numerically stable, specific coding practices must be followed.

Details

1. Automatic Differentiation and `advector`**:** Parameters and intermediate calculations involving them are treated as `advector` objects. RTMB "records" all mathematical operations to compute derivatives automatically. If a function strips these attributes or is not differentiable, the gradient calculation will fail.

2. Avoid Discrete Branching: Standard R conditional statements like `if (x > 0)` or `ifelse()` based on parameter values do not provide derivatives.

- **Problem:** They create "jumps" in the likelihood surface.
- **Solution:** Use smooth approximations. For example, use `fabs` instead of `abs`, or `log_mix` for mixture logic.

3. Numerical Stability: Computations in log-space are preferred to prevent overflow or underflow. Use the following stable utilities instead of raw algebraic expressions:

- log_sum_exp: For summing probabilities in log-space.
- log1p_exp: For log(1 + exp(x)).
- inv_logit: For mapping real numbers to probabilities.

4. Vectorization vs. Loops:

- **Vectorization:** Highly recommended for performance. Standard R vectorized arithmetic (+, -, *, /, log, exp) works seamlessly with AD.
- **Loops:** Standard for loops are safe as long as the operations inside are differentiable.
- **Avoid apply:** Functions like apply, sapply, or lapply may sometimes strip AD attributes and should be replaced with vectorized operations or explicit loops.

5. Matrix Operations: Use specialized functions for matrix algebra to maintain efficiency:

- quad_form_chol: For efficient quadratic forms.
- log_det_chol: For log-determinants via Cholesky factors.

See Also

math_functions, distributions, rtmb_code

---

rtmb_table *RTMB-based Contingency Table Analysis (Chi-squared Test)*

Description

'rtmb_table' performs a chi-squared test of independence between two categorical variables. It provides both classic (frequentist) Pearson chi-squared tests and Bayesian multinomial-style models.

Usage

```
rtmb_table(
  x,
  y = NULL,
  data = NULL,
  correct = TRUE,
  prior = prior_flat(),
  fixed = NULL,
  WAIC = FALSE,
  ...
)
```

Arguments

x	Variable name, formula, table, or matrix.
y	Variable name (optional if x is a formula).
data	A data frame.
correct	Logical; if TRUE, apply Yates' continuity correction for 2x2 classic analyses.
prior	Prior specification (Bayesian mode). Default is 'prior_flat()'.
fixed	Optional named list of fixed values for specific parameters.
WAIC	Logical; if TRUE, add pointwise 'log_lik' to the generate block for WAIC.
...	Additional arguments.

Value

An 'RTMB_Model' object.

Examples

```
# Classic chi-squared test
rtmb_table(skill, cond, data = debate)$classic()
rtmb_table(table(debate$skill, debate$cond))$classic()
```

---

rtmb_ttest *RTMB-based Bayesian two-sample t-test wrapper function*

Description

Performs a Bayesian or Frequentist two-sample t-test using RTMB.

Usage

```
rtmb_ttest(
  x,
  y = NULL,
  data = NULL,
  r = 0.707,
  paired = FALSE,
  ID = NULL,
  y_range = NULL,
  prior = prior_flat(),
  init = NULL,
  fixed = NULL,
  var.equal = TRUE,
  missing = c("listwise", "fiml"),
  WAIC = FALSE,
  ...
)
```

Arguments

x	Numeric vector of responses for group 1, a formula (e.g., 'y ~ group'), or a column name (unquoted) if 'data' is provided.
y	Numeric vector of responses for group 2, or a column name (unquoted) if 'data' is provided. Required if 'x' is not a formula.
data	Data frame containing the variables.
r	Numeric; Cauchy prior scale for the effect size (delta). Default is 0.707.
paired	Logical; whether to perform a paired t-test.
ID	Character; name of the ID variable for paired t-tests (required for formula input with paired = TRUE).
y_range	Theoretical minimum and maximum values of the response variable as a vector c(min, max). Required when using weakly informative priors.
prior	An object of class '"rtmb_prior"'. Use 'prior_flat()' for no prior, 'prior_normal()' for default normal/exponential priors, or 'prior_weak()' for weakly informative Bayesian inference. Default is 'prior_flat()'.
init	List of initial values.
fixed	Optional named list of fixed values for specific parameters.
var.equal	Logical; whether to assume equal variances. Default is TRUE.
missing	Missing value handling strategy: "listwise".
WAIC	Logical; if TRUE, add pointwise 'log_lik' to the generate block for WAIC.
...	Additional arguments.

Details

For classic inference, heteroscedastic two-sample t-tests use the same RTMB Satterthwaite machinery as 'optimize(marginal = ..., df_method = "satterthwaite")'. The result is model-based and reproducible from the printed model code. This corresponds to the Welch-type unequal-variance t-test, but the degrees of freedom are computed by the package's internal Satterthwaite procedure rather than by a separate closed-form formula. When 'prior_jzs()' is combined with 'var.equal = FALSE', BayesRTMB uses a Welch-style JZS extension: the effect size 'delta' is an explicit parameter with a Cauchy prior, and the group mean difference is scaled by the root-mean-square of the two group standard deviations.

Value

An 'RTMB_Model' object.

Examples

```
# Simulate two-sample data with a true effect size
set.seed(123)
y1 <- rnorm(30, mean = 0.5, sd = 1)
y2 <- rnorm(30, mean = 0.0, sd = 1)

# Fit the Bayesian two-sample t-test model
```

```
# r = 0.707 is the standard scale for the Cauchy prior on the effect size
fit_ttest <- rtmb_ttest(y1, y2, prior = prior_jzs(r = 0.707))


# MCMC sampling (chains and iterations reduced for faster execution)
mcmc_ttest <- fit_ttest$sample(sampling = 500, warmup = 500, chains = 2)
mcmc_ttest$summary()
```

`rtmb_wrappers` *Common Features and Arguments of RTMB Wrapper Functions*

Description

The RTMB wrapper functions (‘rtmb_lm‘, ‘rtmb_glm‘, ‘rtmb_glmer‘, ‘rtmb_fa‘, etc.) share a unified interface designed to make Bayesian and Frequentist inference accessible through familiar R formulas and standard model specifications.

Details

All wrapper functions in this package are built upon the same core engine. This ensures that regardless of the model type, the workflow for estimation, summary, and expansion remains consistent.

1. Unified Inference Methods: Every model object returned by a wrapper function provides the following methods:

- `$classic()`: Performs standard frequentist estimation (MLE/REML) using ‘prior_flat()‘. Supports robust standard errors and Satterthwaite/Between-Within degrees of freedom approximations.
- `$optimize()`: Performs Maximum A Posteriori (MAP) estimation (Empirical Bayes).
- `$sample()`: Performs MCMC sampling (NUTS via tmbstan) for full Bayesian inference.
- `$variational()`: Performs Variational Inference (ADVI) for fast posterior approximation.

2. Prior API and Regularization: You can specify the prior distribution using the ‘prior‘ argument:

- `prior_flat()`: No additional prior (default). Required for `classic()` inference.
- `prior_normal()`: Manual normal/exponential priors for parameters.
- `prior_weak()`: Weakly informative priors based on `y_range`.
- `prior_rhs()`: Regularized Horseshoe prior for continuous shrinkage.
- `prior_ssp()`: Spike-and-Slab prior for sparse variable selection.

Note: When using ‘prior_weak()‘, ‘prior_rhs()‘, or ‘prior_ssp()‘, you must specify `y_range = c(min, max)` *to let the model set appropriate global scales for the priors.*

3. Model Comparison and Bayes Factors: Model comparison is performed exclusively via Bridge Sampling. To compare a full model against a nested null model, use the `$bayes_factor(fixed =`

`list(...))` method on an MCMC fit object. For example, `fit$bayes_factor(fixed = list(b = 0))` tests the hypothesis that the fixed effect b is exactly zero.

4. Standard Errors and Degrees of Freedom: The `$classic()` and `$optimize()` methods provide advanced summary options:

- `se_method`: Calculate ‘"robust"‘ (Huber-White) or ‘"cluster-robust"‘ standard errors by providing a cluster variable.
- `df_method`: Use ‘"satterthwaite"‘ for mixed models or ‘"bw"‘ (Between-Within) for multilevel correlation models to obtain accurate p-values and confidence intervals.

5. Fixed parameters: Use `fixed = list(...)` during model construction to fix model parameters to specified values. For example, `fixed = list(delta = 0)` fixes the parameter `delta` to zero. This same syntax is used in `$bayes_factor()` to define the restricted model.

---

r_hat *Calculate Rank-normalized Split-R-hat*

Description

Calculate Rank-normalized Split-R-hat

Usage

```
r_hat(sims)
```

Arguments

sims	A matrix of samples (iterations x chains).

Value

A numeric value.

---

safe_rtmb_model *Safe RTMB model construction (with error message translation)*

Description

Wraps ‘rtmb_model‘ to translate cryptic error messages generated during ‘MakeADFun‘ execution (originating from C++/RTMB) into user-friendly hints.

Usage

```
safe_rtmb_model(data, code, par_names = list(), init = NULL, view = NULL)
```

Arguments

data	A list of data that has passed ‘validate_data‘.
code	Code blocks for likelihood and priors defined with ‘model_code()‘.
par_names	A list of variable names corresponding to the dimensions of each parameter (optional).
init	A list or numeric vector of initial values (optional).
view	Character vector of parameter names to be displayed preferentially in summary outputs (optional).

Value

Returns an ‘rtmb_model‘ object (R6 class instance) upon successful compilation.

---

simple_effects *Calculate Simple Effects*

Description

Calculate the effect of a focal variable at different levels of a moderator. For categorical focal variables, it calculates pairwise differences (contrasts). For continuous focal variables, it calculates the slope (simple slopes).

Usage

```
simple_effects(fit, effect, prob = 0.95, sd_multiplier = 1, ...)
```

Arguments

fit	Model fit object (e.g., ‘map_fit‘, ‘mcmc_fit‘).
effect	Character string of the interaction (e.g., "A:B"). The first variable is the focal variable.
prob	Probability for the credible/confidence interval (default is 0.95).
sd_multiplier	Multiplier for SD for continuous moderators (default is 1).
...	Additional arguments.

Value

A ‘ce_simple‘ object (data frame) containing the estimated effects and their credible intervals.

Examples

```
data(debate, package = "BayesRTMB")
fit <- rtmb_lm(sat ~ talk * perf, data = debate)
map_fit <- fit$optimize()
# Effect of talk at different levels of performance
se <- simple_effects(map_fit, effect = "talk:perf")
print(se)
```

simple_effects.mcmc_fit *Simple effects for MCMC fit objects*

Description

Simple effects for MCMC fit objects

Usage

```
## S3 method for class 'mcmc_fit'
simple_effects(fit, effect, prob = 0.95, sd_multiplier = 1, ...)
```

Arguments

fit	An object of class ‘MCMC_Fit‘.
effect	Interaction term (e.g., "A:B").
prob	Probability for credible intervals.
sd_multiplier	Multiplier for SD for continuous moderators.
...	Additional arguments.

softmax *Softmax function*

Description

Softmax function

Usage

```
softmax(x)
```

Arguments

x	A numeric vector.

Value

A numeric vector of softmax probabilities.

sort_loadings *Sort and display factor loadings neatly*

Description

Sort and display factor loadings neatly

Usage

```
sort_loadings(loadings, cutoff = 0, round_digits = 3)
```

Arguments

loadings	Matrix, data frame, or list of factor loadings (if list, the first element is used)
cutoff	Absolute loadings below this value will be displayed as blank (default is 0.0)
round_digits	Number of decimal places to display (default is 3)

Value

Sorted loading matrix (returned invisibly, allowing assignment to a variable)

squared_distance *Squared Euclidean distance*

Description

Squared Euclidean distance

Usage

```
squared_distance(x, y, eps = 1e-08)
```

Arguments

x	A numeric vector.
y	A numeric vector.
eps	A small value added for numerical stability (default is 1e-8).

Value

The squared Euclidean distance between x and y.

stz_basis *stz basis function*

Description

stz basis function

Usage

```
stz_basis(K)
```

Arguments

K	A numeric value

Value

A transformation matrix

summary.ce_rtmb *Summary method for ce_rtmb class*

Description

Summary method for ce_rtmb class

Usage

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

Arguments

object	An object of class ce_rtmb
...	Additional arguments.

sum_to_zero *Sum-to-zero transformation*

Description

Transforms a vector of length K-1 to a vector of length K that sums to zero using an orthogonal basis matrix.

Usage

```
sum_to_zero(x)
```

Arguments

x	A numeric vector of length K-1.

Value

A numeric vector of length K whose elements sum to zero.

test_info *Calculate Test Information Function*

Description

Calculate Test Information Function

Usage

```
test_info(x, ...)
```

Arguments

x	An object of class RTMB_Fit_Base
...	Additional arguments.

Examples

```
fit <- rtmb_irt(data = BigFive[, 1:5], model = "2PL", type = "ordered")
map_fit <- fit$optimize()
ti <- test_info(map_fit)
plot(ti)
```

to_centered_matrix *Vector to centered matrix (RTMB compatible)*

Description

Vector to centered matrix (RTMB compatible)

Usage

```
to_centered_matrix(x, R, C)
```

Arguments

x	A numeric vector of length (R-1) * C.
R	Number of rows.
C	Number of columns.

Value

An R x C matrix where each column sums to zero.

to_centered_tri *Vector to centered triangular matrix (RTMB compatible)*

Description

Vector to centered triangular matrix (RTMB compatible)

Usage

```
to_centered_tri(x, R, C)
```

Arguments

x	A numeric vector of appropriate length.
R	Number of rows.
C	Number of columns.

Value

An R x C matrix with column-wise sum-to-zero constraints on lower elements.

to_long *Convert Wide Data to Long Format*

Description

A highly intuitive wrapper around `stats::reshape` designed for psychological research. It converts data from wide format to long format by identifying within-subjects factors.

Usage

```
to_long(data, within = NULL, label = NULL, value = "Value", id = NULL)
```

Arguments

`data`	A data frame in wide format.
`within`	The columns to gather into long format. Can be: • A character vector of column names. • A range string like `"time1:time4"`. • A prefix string like `"time"` (matches all columns starting with "time").
`label`	The name for the new column that will contain the level names (e.g., "Time"). If NULL, it defaults to the prefix used in `within` or "Condition".
`value`	The name for the new column that will contain the measurement values. Default is "Value".
`id`	The identifier columns that should be repeated for each row. If NULL (default), all columns NOT specified in `within` are treated as IDs.

Value

A data frame in long format.

to_lower_tri *Vector to lower triangular matrix (RTMB compatible)*

Description

Vector to lower triangular matrix (RTMB compatible)

Usage

```
to_lower_tri(x, M, D)
```

Arguments

x	A numeric or advector.
M	Number of rows.
D	Number of columns.

Value

An M x D lower triangular matrix.

---

to_wide *Convert Long Data to Wide Format*

Description

A user-friendly wrapper around `stats::reshape` to convert data from long format back to wide format.

Usage

```
to_wide(data, within = "Condition", value = "Value", id = NULL)
```

Arguments

data	A data frame in long format.
within	The name of the column containing within-subjects factor levels (e.g., "Condition").
value	The name of the column containing measurement values (e.g., "Value").
id	The identifier columns (e.g., "Subject", "Group"). If NULL, inferred as all columns except `within` and `value`.

Value

A data frame in wide format.

training *Social Skills Training Data*

Description

A small dataset containing repeated outcome measurements from a social skills training study. The outcome was measured at four time points for participants assigned to a control or intervention condition.

Usage

```
training
```

Format

A data frame with 12 rows and 8 columns:

ID Participant identifier.

time1 Outcome score at the first measurement occasion.

time2 Outcome score at the second measurement occasion.

time3 Outcome score at the third measurement occasion.

time4 Outcome score at the fourth measurement occasion.

a Training condition indicator: 0 = control group, 1 = intervention group.

b Qualitative baseline skill classification.

age Participant age.

Source

Simulated dummy data created by the package author.

transform_code *Transformed Code Wrapper for RTMB*

Description

Transformed Code Wrapper for RTMB

Usage

```
transform_code(expr, env = parent.frame())
```

Arguments

expr	A block of code containing calculations for transformed parameters.
env	Environment to assign to the generated function.

Value

A function taking (dat, par) that returns a named list.

`validate_data` *Pre-validation of data and parameters*

Description

Before passing data to ‘rtmb_model‘, it checks for the presence of R-specific data types (such as data.frame) and missing values, and outputs errors or warnings accordingly.

Usage

```
validate_data(dat_list)
```

Arguments

`dat_list`	A list of data to be passed to the model (usually containing matrices or numeric vectors).

Details

RTMB's automatic differentiation engine requires pure ‘matrix‘ or ‘numeric‘ types. If a user mistakenly passes a ‘data.frame‘, incomprehensible errors occur during the construction of the computation graph. This function catches such issues in advance.

Value

Returns invisible ‘NULL‘ on success. Interrupts execution with ‘stop()‘ or issues ‘warning()‘ if issues are found.

`VB_Fit` *VB fit object*

Description

VB fit object

VB fit object

Details

An R6 class storing posterior samples and related information from Automatic Differentiation Variational Inference (ADVI).

Super class

BayesRTMB::RTMB_Fit_Base -> advi_fit

Public fields

model An 'RTMB_Model' object used for estimation.

fit A 3D array of posterior draws for fixed model parameters.

random_fit A 3D array of posterior draws for random effects, if available.

transform_fit A 3D array of posterior draws for transformed parameters, if available.

generate_fit A 3D array of posterior draws for generated quantities, if available.

transform_dims A list storing dimension information for transformed parameters.

generate_dims A list storing dimension information for generated quantities.

elbo_history A list of numeric vectors storing the Evidence Lower Bound (ELBO) history during optimization for each chain.

laplace Logical; whether Laplace approximation was used to marginalize random effects.

posterior_mean A named numeric vector of posterior mean estimates.

ELBO A numeric vector of final ELBO values for each chain.

rel_obj_vals A numeric vector of final relative objective tolerance values for each chain.

best_chain Integer; the index of the chain with the maximum ELBO.

mu_history Matrix of the parameter trajectory from the final window.

Methods

Public methods:

- VB_Fit$get_point_estimate()
- VB_Fit$new()
- VB_Fit$print()
- VB_Fit$draws()
- VB_Fit$summary()
- VB_Fit$plot_elbo()
- VB_Fit$plot_trajectory()
- VB_Fit$transformed_draws()
- VB_Fit$generated_quantities()
- VB_Fit$WAIC()
- VB_Fit$diagnose()
- VB_Fit$clone()

Method `get_point_estimate()`**:** Get point estimate for a target parameter.

Usage:

```
VB_Fit$get_point_estimate(target, chains = NULL, best_chains = NULL)
```

Arguments:

`target` Target parameter name.

`chains` Numeric vector of chains to include. If NULL, all chains are used.

`best_chains` Integer; number of best chains to retain based on ELBO.

Returns: Matrix, array, vector, or scalar point estimate.

Method `new()`**:** Create a new 'VB_Fit' object.

Usage:

```
VB_Fit$new(
  model,
  fit,
  random_fit,
  elbo_history,
  laplace,
  posterior_mean,
  ELBO,
  rel_obj_vals,
  best_chain,
  mu_history
)
```

Arguments:

`model` An 'RTMB_Model' object.

`fit` A 3D array of parameter draws.

`random_fit` A 3D array of random effect draws.

`elbo_history` A list of numeric vectors of ELBO values for each chain.

`laplace` Logical; indicates if Laplace approximation was used.

`posterior_mean` A named numeric vector of posterior means.

`ELBO` A numeric vector of final ELBO values for each chain.

`rel_obj_vals` A numeric vector of final relative objective tolerance values for each chain.

`best_chain` Integer; the index of the chain with the maximum ELBO.

`mu_history` Matrix of the parameter trajectory from the final window.

Method `print()`**:** Print a brief summary of the fitted object.

Usage:

```
VB_Fit$print(...)
```

Arguments:

`...` Additional arguments passed to the 'summary' method.

Returns: The object itself, invisibly.

Method `draws()`**:** Extract posterior draws for selected parameters.

Usage:

```
VB_Fit$draws(
  pars = NULL,
  chains = NULL,
  best_chains = NULL,
```

```
  inc_random = FALSE,
  inc_transform = TRUE,
  inc_generate = TRUE,
  best_only = FALSE
)
```

Arguments:

`pars` Character or numeric vector specifying the names or indices of parameters to extract. If NULL, all available parameters are extracted.

`chains` Numeric vector specifying the chains to extract. If NULL, draws from all chains are returned.

`best_chains` Integer; number of best chains to retain based on ELBO. If supplied, chains with the highest ELBO are retained.

`inc_random` Logical; whether to include random effects in the output. Default is FALSE.

`inc_transform` Logical; whether to include transformed parameters in the output. Default is TRUE.

`inc_generate` Logical; whether to include generated quantities in the output. Default is TRUE.

`best_only` Logical; whether to extract only from the chain with the maximum ELBO. Default is FALSE unless explicitly requested.

Returns: A 3D array of posterior draws '[iterations, chains, parameters]'.

Method `summary()`**:** Summarize posterior draws.

Usage:

```
VB_Fit$summary(
  pars = NULL,
  max_rows = 10,
  digits = 2,
  inc_random = FALSE,
  inc_transform = TRUE,
  inc_generate = TRUE
)
```

Arguments:

`pars` Character or numeric vector specifying the names or indices of parameters to summarize. If NULL, all available parameters are summarized.

`max_rows` Integer; maximum number of rows to print in the summary table. Default is 10.

`digits` Integer; number of decimal places to print. Default is 2.

`inc_random` Logical; whether to include random effects in the summary. Default is FALSE.

`inc_transform` Logical; whether to include transformed parameters in the summary. Default is TRUE.

`inc_generate` Logical; whether to include generated quantities in the summary. Default is TRUE.

Returns: A data frame containing the summarized posterior statistics.

Method `plot_elbo()`**:** Plot the ELBO history to diagnose convergence.

Usage:

```
VB_Fit$plot_elbo(tail_n = 1000, ests = NULL, type = "l", ...)
```

Arguments:

`tail_n` Integer; the number of recent iterations to plot. If NULL, plots the entire history. Default is 2000.

`ests` Character string '"best"', numeric vector of estimate indices (e.g., 'c(1, 3)'), or 'NULL' to plot all. Default is 'NULL'.

`type` Character string; the type of plot. Default is "l" (lines).

`...` Additional arguments passed to the 'plot' function.

Returns: The object itself, invisibly.

Method `plot_trajectory()`**:** Plot the parameter trajectory from the final optimization window.

Usage:

```
VB_Fit$plot_trajectory(pars = NULL, type = "l", ...)
```

Arguments:

`pars` Character vector specifying the names of parameters to plot. If NULL, plots all available parameters.

`type` Character string; the type of plot. Default is "l" (lines).

`...` Additional arguments passed to the 'matplot' function.

Returns: The object itself, invisibly.

Method `transformed_draws()`**:** Compute transformed parameters from posterior draws.

Usage:

```
VB_Fit$transformed_draws(tran_fn = NULL)
```

Arguments:

`tran_fn` An optional user-supplied function that takes data and parameter lists to return transformed quantities.

Returns: The 'VB_Fit' object itself, invisibly.

Method `generated_quantities()`**:** Compute generated quantities from posterior draws.

Usage:

```
VB_Fit$generated_quantities(code)
```

Arguments:

`code` An 'rtmb_code({ ... })' or '{ ... }' block containing the logic to be calculated using posterior samples.

Returns: The 'VB_Fit' object itself (invisibly). Results are appended to the 'generate_fit' field.

Method `WAIC()`**:** Compute WAIC from pointwise generated log likelihood.

Usage:

```
VB_Fit$WAIC(...)
```

Arguments:

`...` Additional arguments passed to 'draws()', such as 'chains' or 'best_chains'.

Returns: A 'waic_BayesRTMB' object.

Method `diagnose()`**:** Run basic diagnostics for the variational fit.

Usage:

```
VB_Fit$diagnose(...)
```

Arguments:

`...` Additional arguments passed to 'diagnose_vb_fit()'.

Returns: A 'diagnose_BayesRTMB' object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
VB_Fit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index