Package: mixeff (via r-universe)

July 11, 2026

Title Audit-First Mixed-Effects Models via the 'mixeff-rs' Rust Crate

Version 0.2.0

Description An R wrapper for the 'mixeff-rs' Rust crate, providing linear and generalized linear mixed-effects model fitting via lme4-style formulas. The wrapper is audit-first: every printed claim traces back to a versioned JSON artifact produced by the Rust compiler, and the package refuses to fabricate inference results that the engine cannot certify. See the package vignettes for migration from 'lme4' and the demystification surface for random-effects syntax.

License MIT + file LICENSE

URL https://bbuchsbaum.github.io/mixeff/, https://github.com/bbuchsbaum/mixeff

BugReports https://github.com/bbuchsbaum/mixeff/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager, >= 1.78.0), rustc (>= 1.78.0), GNU make

Depends R (>= 4.2)

Imports jsonlite, Matrix, rlang, stats

Suggests broom, broom.mixed, generics, ggplot2, emmeans (>= 1.4), estimability, jsonvalidate, knitr, lme4, lmerTest, MASS, rextendr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/website albersdown

Config/pak/sysreqs make libclang-dev

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

Date/Publication 2026-07-11 01:45:38 UTC

RemoteUrl https://github.com/bbuchsbaum/mixeff

RemoteRef HEAD

RemoteSha fb76713b0bad59020bcd3b58979c3546852eb317

Contents

`anova.mm_glmm` *Analysis of deviance for GLMMs*

Description

With two or more fitted models, `anova()` performs a sequential likelihood-ratio comparison (like `anova(glmer1, glmer2)`). For a single model, fixed-effect tests are routed to `drop1()` (term LRTs), `summary()` (Wald z), or `contrast()` (custom Wald contrasts), which the GLMM contract supports directly.

Usage

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

Arguments

`object`	A fitted `mm_glmm`.
`...`	Additional fitted models to compare.

Value

An `mm_model_comparison` object (multi-model case).

as_json *Serialize a mixeff spec or fit to JSON*

Description

`as_json()` returns a JSON string that carries the parsed object's public R-side state and the raw compiler artifact JSON. `saveRDS()` / `readRDS()` remains the primary persistence path for fitted objects; `revive()` restores process-local caches after deserialization.

Usage

```
as_json(x, pretty = FALSE, ...)

## S3 method for class 'mm_compiled'
as_json(x, pretty = FALSE, ...)
```

Arguments

`x`	A compiled `mm_spec` or fitted `mm_fit`.
`pretty`	Logical; pretty-print JSON when `TRUE`.
`...`	Reserved for future methods.

Value

A length-one character string containing JSON.

audit *Audit a compiled model spec or fitted model*

Description

`audit()` returns the user-facing audit report attached to an `mm_spec` (pre-fit) or an `mm_fit` (post-fit). The text is rendered by the upstream Rust crate (the `mixedmodels.model_audit_report` schema's `Display` impl) — Rust authors the wording, R formats nothing. Routing every printed audit line through the upstream renderer is what enforces the R9 "no advice creep" contract: drift in scope notes / tone is visible in one place rather than scattered across R formatters.

Usage

```
audit(object, ...)

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

## S3 method for class 'mm_compiled'
audit(object, ...)
```

Arguments

object	An mm_spec produced by compile_model() or an mm_fit from lmm()/glmm().
...	Reserved for future methods.

Details

audit() accepts an mm_spec from compile_model() or an mm_fit from lmm()/glmm() and emits the report sections (Requested Model, Model State, Fixed/Random Effects, Information Budget, Dependence Paths, Parameterization Trace, Effective Covariance, Policy Recommendations, Optimizer, Inference, Diagnostics). Sections that depend on a fit (Optimizer / Inference) report not applicable before fitting on a pre-fit spec.

Value

An object of class mm_audit carrying:

text the rendered report text (a single character string, newline-separated)

summary_text the compact report rendered by the upstream ModelAuditReport::render_summary (Audit Summary plus the Requested Model section)

design_audit the parsed design_audit field from the CompiledModelArtifact (random-term audits, fixed-effect rank, covariance kernel graph, ...) — NULL on uncompilable formulas

report the parsed upstream ModelAuditReport v2, including Rust-authored random_term_cards for downstream explanation verbs

random_term_cards the report's per-random-term cards, copied to the top level for convenient inspection

cross_card_constraints report-level constraints between random-term cards

diagnostics the parsed report diagnostics, falling back to artifact diagnostics when needed

print.mm_audit defaults to the compact upstream-rendered summary in summary_text. Use print(x, full = TRUE) for the complete upstream report stored in text.

Errors

Raises an mm_schema_error if the supplied object does not carry a parsed artifact with the expected schema header.

See Also

compile_model().

Examples

```
## Not run:
df <- data.frame(
  y       = rnorm(20),
  x       = rnorm(20),
  subject = factor(rep(letters[1:5], each = 4))
)
```

```
audit(compile_model(y ~ x + (1 + x | subject), df))

## End(Not run)
```

`audit_design` *Deprecated alias for* `audit()`

Description

`audit_design()` and `audit()` were two names for one operation; the surface collapsed to `audit()` for 0.2.0. This alias forwards with a deprecation note and will be removed in a future release.

Usage

```
audit_design(spec)
```

Arguments

`spec`	An `mm_spec` or `mm_fit`; see `audit()`.

Value

See `audit()`.

`bootstrap_control` *Fixed-effect bootstrap control*

Description

Fixed-effect bootstrap control

Usage

```
bootstrap_control(
  nsim = 999L,
  seed = NULL,
  failed_refit_policy = c("exclude", "count_extreme", "abort")
)
```

Arguments

`nsim`	Requested bootstrap replicate count.
`seed`	Optional integer seed. `NULL` leaves the Rust RNG seed unspecified and records that state in row details.
`failed_refit_policy`	How failed refits are accounted for. Stable Rust wire labels are `"exclude"`, `"count_extreme"`, and `"abort"`.

Value

A list used by `contrast(..., method = "bootstrap")`.

`changes` *Show requested, effective, and fitted model-state changes*

Description

`changes()` summarizes the transitions recorded in the compiler artifact: requested formula to effective formula, design-time reductions or covariance transitions, and fitted covariance rank/status from the optimizer certificate pass.

Usage

```
changes(object, ...)

## S3 method for class 'mm_compiled'
changes(object, ...)
```

Arguments

`object`	A compiled `mm_spec` or fitted `mm_fit`.
`...`	Reserved for future methods.

Value

An `mm_change_log` object with a data-frame `table` and the raw artifact fragments used to build it.

`compare` *Compare fitted mixeff models*

Description

`compare()` is the namespace-qualified model-comparison front door. For LMMs it reports likelihood, information criteria, and asymptotic likelihood-ratio comparisons. REML fits are refit by ML when `refit_for_comparison = "auto"` or `"ml"`; `"error"` refuses that comparison.

Usage

```
compare(object, ...)

## S3 method for class 'mm_lmm'
compare(
  object,
  ...,
  target = c("fixed_effects", "random_effects", "prediction"),
  method = c("auto", "lrt", "bootstrap", "aic"),
  refit_for_comparison = c("auto", "error", "ml"),
  nsim = 0L,
  seed = NULL
)
```

Arguments

`object`	A fitted `mm_lmm`.
`...`	Additional fitted `mm_lmm` objects.
`target`	Comparison target label.
`method`	`"auto"` / `"lrt"` for asymptotic likelihood-ratio rows, `"aic"` for information criteria only, or `"bootstrap"` for a small parametric-bootstrap LRT when `nsim > 0`.
`refit_for_comparison`	How to handle REML fits.
`nsim`	Number of bootstrap simulations for `method = "bootstrap"`.
`seed`	Optional bootstrap seed.

Value

An `mm_model_comparison` object with a data-frame `table`.

`compare_covariance` *Compare covariance parameterizations for current random terms*

Description

`compare_covariance()` is a compact alternate view of the same upstream random-term cards used by `explain_model()` and `random_options()`. For each current random-term card, it lays out the full, diagonal, and scalar covariance families without ranking them.

Usage

```
compare_covariance(spec)
```

Arguments

spec	An mm_spec from compile_model() or, in later phases, an mm_fit.

Value

An object of class mm_compare_covariance with a table data frame and the upstream cards it was derived from.

---

compile_model *Compile a mixed-effects model spec without fitting*

Description

compile_model() parses the formula, runs the upstream semantic-IR / design-audit pipeline against the supplied data, and returns an mm_spec object — the audit-first analogue of the design-only step in base lm()'s model.frame() / model.matrix() chain. Nothing is optimized; nothing is fitted. audit(), explain_model(), random_options(), and (in Phase 1.E) lmm() all consume the same artifact.

Usage

```
compile_model(formula, data)
```

Arguments

formula	A two-sided lme4-style formula, e.g. y ~ x + (1 + x | subject).
data	A data.frame whose columns include every variable named in formula. Variables with missing values raise an mm_data_error; pass na.omit(data) explicitly if that is what you want.

Details

The compiled artifact is the structured truth: every print, summary, and audit verb in mixeff reads back from it rather than re-deriving meaning from formula text. R formats; Rust authors wording (PRD §9.6).

Phase 1 compile scope: returns a populated mm_spec with the JSON artifact attached. explain_model(), random_options(), and compare_covariance() render random-effects guidance from upstream random-term cards; the fit driver (lmm()) lands in 1.E.

Value

An object inheriting from mm_spec and containing:

call the matched call

formula the input formula

vars character vector of variables read from data

`model_frame` the data columns used to compile the artifact, retained so prefit audit views can evaluate nearby formula spellings

`artifact` parsed JSON artifact (the `mixedmodels.compiled_model_artifact` v1 schema)

The raw artifact JSON is attached as `attr(spec$artifact, "raw_json")` so the post-compile FFI calls (e.g., the internal `mm_audit_report_text` primitive) can round-trip without re-encoding.

Errors

Raises typed conditions (all inheriting from `mm_condition`):

- `mm_formula_error` — formula is not a two-sided R formula or fails parsing.
- `mm_data_error` — `data` is not a data.frame, refers to unknown variables, contains NAs in design columns, or has an unsupported column type.
- `mm_schema_error` — the artifact JSON returned by Rust does not match the wrapper's known schema set.

See Also

`audit()` for the printed audit report.

Examples

```
## Not run:
df <- data.frame(
  y       = rnorm(20),
  x       = rnorm(20),
  subject = factor(rep(letters[1:5], each = 4))
)
spec <- compile_model(y ~ x + (1 + x | subject), df)
audit(spec)

## End(Not run)
```

---

confint.mm_glmm *Confidence intervals for fixed effects of a mixeff GLMM*

Description

Asymptotic Wald intervals (`estimate +/- z * SE`) built from the Rust fixed-effect inference table. Profile and bootstrap intervals are not certified for GLMMs by the upstream contract and are refused with a typed reason rather than approximated.

Usage

```
## S3 method for class 'mm_glmm'
confint(
  object,
  parm,
  level = 0.95,
  method = c("asymptotic", "wald", "profile", "bootstrap"),
  ...
)
```

Arguments

object	A fitted mm_glmm.
parm	Optional fixed-effect names or indices; defaults to all.
level	Confidence level.
method	"asymptotic" (the default; the package-wide name for the closed-form Wald interval) or its synonym "wald".
...	Unused.

Value

An mm_confint matrix of lower/upper bounds.

---

contrast.mm_glmm *Contrast fixed effects*

Description

Note: this is not emmeans::contrast. contrast() is mixeff's fixed-effect contrast front door. R validates the contrast matrix shape, then asks Rust to evaluate estimability, method prerequisites, standard errors, degrees of freedom, statistics, p-values, reliability, and unavailable reasons.

Usage

```
## S3 method for class 'mm_glmm'
contrast(fit, L, rhs = 0, method = c("asymptotic", "wald"), ...)

contrast(
  fit,
  L,
  rhs = 0,
  method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "asymptotic",
    "boundary_lrt", "none"),
  bootstrap = NULL,
  ...
)
```

```
## S3 method for class 'mm_lmm'
contrast(
  fit,
  L,
  rhs = 0,
 method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "asymptotic",
    "boundary_lrt", "none"),
  bootstrap = NULL,
  ...
)
```

Arguments

fit	A fitted mm_lmm.
L	A numeric contrast vector or matrix with one column per fixed effect.
rhs	Numeric right-hand side, recycled to the number of contrasts.
method	Requested inference method.
...	Reserved for future methods.
bootstrap	Optional bootstrap_control() object for method = "bootstrap".

Details

For mm_glmm fits, contrasts use an asymptotic Wald z-test built from the stored fixed-effect covariance (the GLMM contract does not provide finite-sample df), so method accepts only "asymptotic" (alias "wald").

Value

An mm_contrast object with a data-frame table. The estimate column is the tested difference, L beta_hat - rhs.

---

df_for_contrast *Degrees of freedom for a contrast*

Description

Degrees of freedom for a contrast

Usage

```
df_for_contrast(
  fit,
  L,
 method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "asymptotic", "none"),
  ...
```

```
)

## S3 method for class 'mm_lmm'
df_for_contrast(
  fit,
  L,
 method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "asymptotic", "none"),
  ...
)
```

Arguments

`fit`	A fitted `mm_lmm`.
`L`	A contrast vector or matrix.
`method`	Requested degrees-of-freedom method.
`...`	Reserved for future methods.

Value

An `mm_df_for_contrast` object with `$table` (one row per contrast: `contrast`, `df`, `method`, `requested_method`, `reason`), `$df` (the named numeric vector), and `$method`. When the method is `"none"` or the engine refuses, `df` is `NA` and `reason` records why.

`diagnostics` *Inspect mixeff diagnostics and fit status*

Description

`diagnostics()` returns the structured diagnostics carried by a compiled spec or fitted model artifact. `fit_status()` is the compact status string recorded by the optimizer certificate for fitted models.

Usage

```
diagnostics(fit, severity = NULL, stage = NULL, ...)

## S3 method for class 'mm_compiled'
diagnostics(fit, severity = NULL, stage = NULL, ...)

fit_status(fit, ...)

## S3 method for class 'mm_fit'
fit_status(fit, ...)

## S3 method for class 'mm_compiled'
fit_status(fit, ...)
```

Arguments

fit	A compiled mm_spec or fitted mm_fit.
severity	Optional character vector used to filter diagnostics by severity.
stage	Optional character vector used to filter diagnostics by stage.
...	Reserved for future methods.

Value

diagnostics() returns an mm_diagnostics object containing the raw diagnostic list and a data-frame view. fit_status() returns a length-one character string.

---

drop1.mm_glmm *Drop one fixed-effect term at a time from a GLMM*

Description

Refits reduced fixed-effect GLMMs (random-effect terms preserved exactly) and compares each to the full fit by asymptotic likelihood-ratio test, mirroring drop1(glmerMod, test = "Chisq").

Usage

```
## S3 method for class 'mm_glmm'
drop1(object, scope = NULL, test = c("none", "Chisq"), ...)
```

Arguments

object	A fitted mm_glmm.
scope	Optional character vector of fixed-effect terms to drop.
test	"Chisq" reports asymptotic LRT rows; "none" reports information criteria only.
...	Reserved for future methods.

Value

An mm_drop1 object.

drop1.mm_lmm *Drop one fixed-effect term at a time*

Description

drop1.mm_lmm() refits reduced fixed-effect models and compares them to the original fit. It is conservative: random-effect terms are preserved exactly, and the reduced formulas are reported in the result table.

Usage

```
## S3 method for class 'mm_lmm'
drop1(
  object,
  scope = NULL,
  test = c("none", "Chisq"),
  refit_for_comparison = c("auto", "error", "ml"),
  ...
)
```

Arguments

object	A fitted mm_lmm.
scope	Optional character vector of fixed-effect terms to drop.
test	Comparison test label. "Chisq" reports asymptotic LRT rows; "none" reports information criteria only.
refit_for_comparison	How to handle REML fits.
...	Reserved for future methods.

Value

An mm_drop1 object.

estimability *Assess contrast estimability*

Description

Routes each requested contrast row through the Rust fixed-effect inference bridge and reports the upstream estimability assessment verbatim. Returned rows carry status (the closed enum from upstream: estimable, not_estimable, aliased, ...), a boolean estimable convenience flag, the contrast rank and requested_rank, and a stable reason populated only when the engine refuses the contrast.

Usage

```
estimability(fit, L = NULL, ...)

## S3 method for class 'mm_lmm'
estimability(fit, L = NULL, ...)
```

Arguments

fit	A fitted mm_lmm.
L	Optional contrast vector or matrix. Defaults to the fixed-effect coefficient basis.
...	Reserved for future methods.

Value

An mm_estimability object.

---

explain_model *Explain the random-effects structure of a compiled model*

Description

explain_model() renders the random-effects guidance surface for an mm_spec returned by compile_model() or, in later phases, an mm_fit. It formats the upstream RandomTermCard and diagnostic payloads; Rust remains the source of truth for per-block English wording and design facts.

Usage

```
explain_model(spec)
```

Arguments

spec	An mm_spec produced by compile_model() or an mm_fit.

Value

An object of class mm_explanation carrying:

text the rendered explanation text

cards the upstream random-term cards

cross_card_constraints report-level constraints between cards

diagnostics the upstream diagnostics used for design notes

report the parsed upstream ModelAuditReport

Errors

Raises an mm_schema_error if spec is not an mm_spec/mm_fit or does not carry a valid compiled artifact.

See Also

compile_model(), audit().

Examples

```
## Not run:
df <- data.frame(
  y = rnorm(20),
  t = rep(0:3, 5),
  s = factor(rep(1:5, each = 4))
)
explain_model(compile_model(y ~ t + (1 | s), df))

## End(Not run)
```

---

fit_handle_alive *Test whether a mixeff fit has a live native handle*

Description

The native handle is a process-local cache. A FALSE result does not mean the fit is unusable: Phase 2 extractors read from the durable artifact and flat R-side payload, and revive() recreates the lazy cache after serialization.

Usage

```
fit_handle_alive(fit, ...)

## S3 method for class 'mm_fit'
fit_handle_alive(fit, ...)
```

Arguments

fit	A fitted mm_fit object.
...	Reserved for future methods.

Value

A length-one logical value.

getME *Extract low-level model components*

Description

getME() provides a small, honest subset of the familiar lme4 extractor. The fixed-effect design ("X"), random-effect design ("Z"), relative covariance factor ("Lambda" / "Lambdat"), grouping factors ("flist"), random coefficient names ("cnms"), response ("y"), fixed coefficients ("beta" / "fixef"), and theta vector ("theta") are rebuilt lazily from the serialized R object.

Usage

```
getME(object, name, ...)

## S3 method for class 'mm_lmm'
getME(object, name, ...)
```

Arguments

object	A fitted mm_lmm object.
name	Component name, or a character vector of names.
...	Reserved for future methods.

Value

The requested component, or a named list for multiple names.

glmm *Fit a generalized linear mixed model*

Description

glmm() validates the R-side family/link request, compiles the model formula, and delegates the numerical fit to the upstream Rust GeneralizedLinearMixedModel. The default method = "pirls_profiled" is the labelled fast-PIRLS path. method = "joint_laplace" uses the upstream labelled joint Laplace route (fast = FALSE, nAGQ = 1) backed by the native dependency-light optimizer in this vendored build.

Usage

```
glmm(
  formula,
  data,
  family,
  random = NULL,
  weights = NULL,
  offset = NULL,
  subset = NULL,
  na.action = na.omit,
  contrasts = NULL,
  method = c("pirls_profiled", "joint_laplace"),
  nAGQ = 1L,
  inference = c("auto", "none", "asymptotic", "bootstrap"),
  control = mm_control(),
  ...
)
```

Arguments

formula	A two-sided lme4-style formula.
data	A data.frame.
family	A supported GLMM family object or family constructor. The certified 1.0 surface is: binomial() with "logit", "probit", or "cloglog" links; poisson() with "log" or "sqrt" links; Gamma() with "log" link; and negative binomial (NB2, "log" link) via mm_negative_binomial() (theta estimated, like lme4::glmer.nb()) or MASS::negative.binomial(theta) (fixed theta).
random	Reserved for the native random-effect constructor path.
weights	Optional prior weights. For binomial models these are trial counts for proportion responses; weights must be positive and finite.
offset	Optional fixed linear-predictor offset; values must be finite.
subset, na.action, contrasts	Reserved for future parity with lmm().
method	GLMM estimation method. "pirls_profiled" is the default fast-PIRLS profiled path. "joint_laplace" requests the labelled joint Laplace route and requires nAGQ <= 1. The joint route tracks the lme4 joint-Laplace reference far more closely than the profiled path on high-baseline models, at a higher optimizer cost; cap that cost with mm_control(max_feval =). The default profiled path is **not** glmer's estimator and its coefficients do not match glmer() exactly; when method is left at its default, glmm() emits an informational notice to that effect (suppress with mm_control(verbose = -1)). Use method = "joint_laplace" for glmer-equivalent estimates.
nAGQ	Number of adaptive Gauss-Hermite quadrature points. 1 is the Laplace setting. Values above 1 are allowed on the profiled path and are rejected for method = "joint_laplace" in the R wrapper.
inference	Requested inference method.

control	A list from mm_control().
...	Reserved for future use.

Details

Optimization runs inside a single native call with no progress output: the pre-fit explanation block (when verbose >= 0) is the last thing printed before the fitted result returns, and the call cannot be interrupted from R. Every optimizer budget is bounded, so fits always terminate; runtime on large problems is governed by mm_control(max_feval =).

Value

An object of class mm_glmm, also inheriting from mm_fit and mm_compiled.

Examples

```
set.seed(1)
df <- data.frame(
  y = rbinom(120, 1, 0.5),
  x = rnorm(120),
  g = factor(rep(seq_len(12), each = 10))
)
fit <- glmm(y ~ x + (1 | g), df, family = binomial(),
            control = mm_control(verbose = -1))
fixef(fit)
# glmer-equivalent (joint Laplace) estimates:
fit_joint <- glmm(y ~ x + (1 | g), df, family = binomial(),
                  method = "joint_laplace",
                  control = mm_control(verbose = -1))
fixef(fit_joint)
```

---

inference_options *Inspect inference methods available for this fit*

Description

inference_options() is the audit verb for fixed-effect inference. It does not run any test; it predicts, from the fit's metadata, which inference methods will succeed on this fit and at what approximate cost. The goal is to remove trial-and-error: a user reading the table can see which routes are immediately available, which will refuse and why, and which require a bootstrap.

Usage

```
inference_options(fit, term = NULL, nsim = 1000L, ...)

## S3 method for class 'mm_lmm'
inference_options(fit, term = NULL, nsim = 1000L, ...)
```

Arguments

fit	A fitted mm_lmm.
term	Optional fixed-effect term name. Reserved for future per-term refinement; currently unused (the table is fit-level).
nsim	Bootstrap replicate count to use when estimating cost. Used only to format the approx_cost column.
...	Reserved for future methods.

Details

Like random_options(), this function does not rank or recommend. There is no "best method" row.

Value

An mm_inference_options object with a table data frame of one row per candidate method.

---

inference_table *Fixed-effect inference table*

Description

Fitted artifacts may carry Rust-owned fixed-effect inference rows. When present, those rows are the source of truth for estimates, standard errors, degrees of freedom, statistics, p-values, methods, status, reliability, and unavailable reasons. Legacy objects without this artifact field fall back to an unavailable table.

Usage

```
inference_table(fit, ...)

## S3 method for class 'mm_lmm'
inference_table(
  fit,
  method = c("auto", "satterthwaite", "kenward_roger", "asymptotic", "none"),
  ...
)
```

Arguments

fit	A fitted mm_lmm.
...	Reserved for future methods.
method	Inference method. "auto" (the default) returns the artifact-cached table that the engine resolved at fit time. Any other value ("satterthwaite", "kenward_roger", "asymptotic", "none") recomputes the table by dispatching one contrast() per fixed-effect term with the requested method, so refusals and reasons are surfaced honestly rather than silently swapped for the auto-resolved row.

Value

An `mm_inference_table`.

`is_singular` *Test whether a fit is singular or reduced-rank*

Description

Test whether a fit is singular or reduced-rank

Usage

```
is_singular(x, tol = 1e-04, ...)

## S3 method for class 'mm_lmm'
is_singular(x, tol = 1e-04, ...)
```

Arguments

`x`	A fitted `mm_lmm`.
`tol`	Reserved for compatibility with lme4's `isSingular()`.
`...`	Reserved for future methods.

Value

A length-one logical value.

`lmm` *Fit a linear mixed-effects model*

Description

`lmm()` is mixeff's Phase 1 linear mixed-model fit driver. It compiles the requested lme4-style formula, emits the same `explain_model()` view that pre-fit audit users see as a message (silence it with `suppressMessages()` or `mm_control(verbose = -1)`), then delegates the numerical fit to the upstream Rust `LinearMixedModel`.

Usage

```
lmm(
  formula,
  data,
  REML = TRUE,
  weights = NULL,
  subset = NULL,
  na.action = NULL,
  contrasts = NULL,
  control = mm_control()
)
```

Arguments

formula	A two-sided lme4-style formula, e.g. y ~ x + (1 + x | subject).
data	A data.frame containing all variables in formula.
REML	Logical; fit by restricted maximum likelihood when TRUE.
weights	Optional positive numeric case weights, either a vector with one value per row or an expression evaluated in data.
subset	Optional expression selecting rows of data, evaluated in data (as in stats::lm()).
na.action	Optional function controlling missing-value handling, applied to the model variables before fitting (e.g. stats::na.omit). The default (NULL) refuses any NA in a model variable with a typed mm_data_error (audit-first: missing-data dropping must be opt-in). Pass na.action = na.omit for lme4's complete-case behaviour.
contrasts	Optional named list of factor contrasts. The engine codes unordered factors with treatment contrasts (contr.treatment) and ordered factors with orthonormal polynomial contrasts (contr.poly), matching lme4/R defaults. A request for any other coding is refused (recode the factor instead).
control	A list from mm_control().

Details

The returned object is deliberately serializable: fixed effects, theta, sigma, likelihood summaries, fitted values, residuals, random effects, and the post-fit compiler artifact are all stored directly on the R object. The native Rust handle is treated as a rebuildable cache, not as the source of truth.

Optimization runs inside a single native call with no progress output: the pre-fit explanation block (when verbose >= 0) is the last thing printed before the fitted result returns, and the call cannot be interrupted from R. Every optimizer budget is bounded, so fits always terminate; runtime on large problems is governed by mm_control(max_feval =).

Value

An object of class mm_lmm, also inheriting from mm_fit and mm_compiled.

Examples

```
set.seed(1)
df <- data.frame(
  y = rnorm(80),
  x = rnorm(80),
  subject = factor(rep(seq_len(20), each = 4))
)
fit <- lmm(y ~ x + (1 | subject), df, control = mm_control(verbose = -1))
fixef(fit)
VarCorr(fit)
summary(fit)
```

`mm_control` *Control mixeff fitting behavior*

Description

`mm_control()` collects small R-side controls for `lmm()` and `glmm()`. `verbose = -1` suppresses the pre-fit `explain_model()` message; non-negative values emit it once before optimization (it travels on the message stream, so `suppressMessages()` and knitr's `message = FALSE` also quiet it).

Usage

```
mm_control(
  verbose = 0L,
  max_feval = NULL,
  optimizer = NULL,
  start = NULL,
  ftol_rel = NULL,
  ftol_abs = NULL,
  xtol_rel = NULL
)
```

Arguments

`verbose`	Integer verbosity level. Use `-1` to suppress the automatic model explanation (and the GLMM estimator notice).
`max_feval`	Optional positive integer capping the optimizer's objective evaluations. Most useful for `glmm()` with `method = "joint_laplace"`, whose native joint optimizer otherwise runs to an engine-chosen budget. `NULL` (default) leaves the engine default in place.
`optimizer`	Optional optimizer name, overriding the driver's automatic choice. One of `"auto"` (default behaviour), `"bobyqa"`, `"newuoa"`, `"cobyla"`, `"pattern_search"`, `"trust_bq"`, or the PRIMA variants (`"prima_bobyqa"`, `"prima_cobyla"`, `"prima_lincoa"`, `"prima_newuoa"`). An unsupported or not-compiled choice raises a typed error rather than silently falling back. `NULL`/`"auto"` keep automatic selection.

`start` Optional numeric warm-start vector for the covariance parameters (theta). Its length must match the model's theta dimension (the engine validates this). `NULL` (default) cold-starts.

`ftol_rel`, `ftol_abs` Optional positive relative/absolute convergence tolerances on the objective. `NULL` keeps the engine default.

`xtol_rel` Optional positive relative convergence tolerance on the optimizer parameters. `NULL` keeps the engine default.

Details

By default the fit driver selects the optimizer and its tolerances automatically (see `optimizer_certificate()` to inspect what ran). The `optimizer`, `start`, and `ftol_*`/`xtol_rel` arguments are a narrow, opt-in escape hatch — for recourse when the default fails to converge, for warm starts, and for explicit tolerance overrides. Any override you supply is recorded in the optimizer certificate, so the fit stays auditable.

Value

A list of class `mm_control`.

See Also

`optimizer_certificate()` to inspect which optimizer ran and whether a caller override was applied.

`mm_formula_manifest` *The wrapper's formula manifest*

Description

A versioned record of what `mixeff` currently supports — formula syntax surface, schema versions per artifact type, and capability flags. The manifest is the wrapper's machine-readable answer to *"what does this build know how to do?"*. Every `mm_fit` object created by future phases will store a snapshot of `mm_formula_manifest()` at construction time so the wrapper's `audit()` verb (Phase 1+) can answer the same question for old fits without consulting the Rust handle.

Usage

```
mm_formula_manifest()
```

Details

Capability flags evolve over phases; gate behavior on flags rather than on package version.

Value

A named list with the following elements:

`mixeff_rust_version` Version of the bundled extendr crate.

`crate_version` Version of the bundled `mixedmodels` upstream crate.

`schema_versions` Named list, one entry per artifact schema the wrapper currently emits or consumes.

`formula_features` Named list with `operators`, `intercept_forms`, `random_term_forms`, and `transformations` — the lme4-style syntax surface.

`capabilities` Named list of logical flags (`parse_formula`, `compile_model`, `audit_design`, `explain_model`, `random_options`, `compare_covariance`, `fit_lmm`, `fit_glmm`, `audit`, `changes`, `diagnostics`, `fit_status`, `parameterization`, `roles`, `as_json`, `simulate`, `inference`, `model_comparison_table`, `fit_summary_payload`, `marginal_quantity_table`, `marginal_quantities`, `verify_convergence`). The `marginal_quantity_table` schema may be available before the corresponding `marginal_quantities` verbs are implemented.

Examples

```
m <- mm_formula_manifest()
m$schema_versions$formula
m$capabilities$parse_formula
```

`mm_grid` *Marginal grids, predictions, means, and comparisons*

Description

These helpers provide a small native marginal-quantities surface for Gaussian LMM fits. They cover the common population-level workflow: construct a reference grid, evaluate fixed-effect predictions, average them into marginal means, and compare those means by simple differences.

Usage

```
mm_grid(fit, specs, by = NULL, at = list(), cov.reduce = mean, ...)

## S3 method for class 'mm_lmm'
mm_grid(fit, specs, by = NULL, at = list(), cov.reduce = mean, ...)

mm_predictions(
  fit,
  grid = NULL,
  specs = NULL,
  by = NULL,
  at = list(),
 method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "asymptotic", "none"),
```

```
  level = 0.95,
  target = c("population"),
  scale = c("response", "link"),
  ...
)

## S3 method for class 'mm_lmm'
mm_predictions(
  fit,
  grid = NULL,
  specs = NULL,
  by = NULL,
  at = list(),
 method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "asymptotic", "none"),
  level = 0.95,
  target = c("population"),
  scale = c("response", "link"),
  ...
)

mm_means(
  fit,
  specs,
  by = NULL,
  at = list(),
  grid = NULL,
 method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "asymptotic", "none"),
  level = 0.95,
  weights = c("equal", "proportional"),
  target = c("population"),
  scale = c("response", "link"),
  ...
)

## S3 method for class 'mm_lmm'
mm_means(
  fit,
  specs,
  by = NULL,
  at = list(),
  grid = NULL,
 method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "asymptotic", "none"),
  level = 0.95,
  weights = c("equal", "proportional"),
  target = c("population"),
  scale = c("response", "link"),
  ...
)
```

```
mm_comparisons(
  fit,
  specs,
  by = NULL,
  at = list(),
  grid = NULL,
  comparison = c("difference", "ratio", "odds_ratio"),
 method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "asymptotic", "none"),
  level = 0.95,
  weights = c("equal", "proportional"),
  target = c("population"),
  scale = c("response", "link"),
  ...
)

## S3 method for class 'mm_lmm'
mm_comparisons(
  fit,
  specs,
  by = NULL,
  at = list(),
  grid = NULL,
  comparison = c("difference", "ratio", "odds_ratio"),
 method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "asymptotic", "none"),
  level = 0.95,
  weights = c("equal", "proportional"),
  target = c("population"),
  scale = c("response", "link"),
  ...
)
```

Arguments

fit	A fitted mm_lmm.
specs	Character vector, or a one-sided formula such as ~ trt or ~ trt | group, naming the displayed marginal dimensions.
by	Optional character vector of grouping variables for marginal summaries or pairwise comparisons.
at	Named list of fixed-predictor values to force in the grid.
cov.reduce	Function used to reduce numeric fixed predictors that are not explicitly gridded.
...	Reserved for future methods.
grid	Optional object returned by mm_grid().
method	Requested inference method, passed to contrast().
level	Confidence level for intervals computed from contrast standard errors.
target	Prediction target. Only "population" is implemented.

scale	Prediction scale. Gaussian LMMs have identical "link" and "response" scales.
weights	Averaging weights for mm_means() and mm_comparisons(). "equal" weights reference-grid cells equally; "proportional" weights cells by observed fixed-factor frequencies.
comparison	Comparison scale. Only "difference" is implemented.

Details

The returned tables use the mixedmodels.marginal_quantity_table row contract. Inference is routed through contrast() so rows retain the same method, status, reliability, estimability, and reason fields as fixed-effect contrasts. Ordinary full-rank LMMs use the versioned mixedmodels.fixed_effect_covariance_m payload for fixed-effect uncertainty; rank-deficient or otherwise uncertified fits surface explicit unavailable status and reasons instead of partial covariance numbers.

Value

mm_grid() returns an mm_grid object. The other helpers return an mm_marginal_quantity object with a contract-shaped table.

---

mm_json_known_schemas *Closed list of schema/version pairs the wrapper understands*

Description

Returns the known-schema table that backs mm_json_negotiate(). New schemas appear here as later phases add artifacts (compiled-model, audit, theta_map, certificate, inference, reproducibility, prediction).

Usage

```
mm_json_known_schemas()
```

Value

A data frame with two character columns: name and version.

See Also

mm_json_negotiate().

Examples

```
mm_json_known_schemas()
```

`mm_json_negotiate` *Negotiate a JSON schema header against what* `mixeff` *supports*

Description

Every artifact crossing the Rust-R bridge carries a header that names a schema (e.g., `formula`, `artifact`, `audit`) and the version of that schema (`v0`, `v1`, ...). `mm_json_negotiate()` validates a header against the closed set of `(schema_name, schema_version)` pairs the current wrapper build understands, and raises a typed `mm_schema_error` on mismatch.

Usage

```
mm_json_negotiate(header)
```

Arguments

`header`	A list with at least `schema_name` and `schema_version` as length-1 character strings. Additional fields (e.g., `crate_version`, `package_version`) are accepted and ignored at this layer; downstream code records them on the `mm_fit` provenance.

Details

This is the *fast-fail* primitive: any code path that consumes a Rust artifact should call `mm_json_negotiate()` before parsing the body, so a version skew between the Rust crate and the R wrapper produces a single clean error rather than a confusing field-by-field decode failure.

Value

Invisibly returns `TRUE` on a clean match.

Errors

Any of the following raise an `mm_schema_error` (also inheriting from `mm_condition` and `error`):

- `header` is not a list, or is missing `schema_name` / `schema_version`, or those fields are not length-1 character.
- `schema_name` is not in the wrapper's known set (see `mm_json_known_schemas()`).
- `schema_name` is known but `schema_version` does not match what the wrapper expects.

The condition object carries the original `header` in its `input` field. (The field is *not* called `header` because rlang reserves that name on conditions for `cnd_header()` formatting.)

See Also

`mm_json_known_schemas()` for the closed set, `mm_formula_manifest()` for the broader capability record.

Examples

```
mm_json_negotiate(list(schema_name = "formula", schema_version = "v0"))
## Not run:
# Raises mm_schema_error:
mm_json_negotiate(list(schema_name = "formula", schema_version = "v99"))

## End(Not run)
```

`mm_lincomb` *Wald inference on a linear combination of fixed effects*

Description

Convenience helper for the common case of testing $H_0 : c^\top \beta = 0$ where `c` is a sparse, named weight vector. The estimate is $c^\top \hat{\beta}$, the standard error is $\sqrt{c^\top V c}$ where `V` is the model's fixed-effect covariance, the statistic is the Wald ratio, and the interval is the symmetric Wald CI at `level`.

Usage

```
mm_lincomb(fit, weights, level = 0.95, method = NULL, ...)

## Default S3 method:
mm_lincomb(fit, weights, level = 0.95, method = NULL, ...)

## S3 method for class 'mm_glmm'
mm_lincomb(fit, weights, level = 0.95, method = NULL, ...)

## S3 method for class 'mm_lmm'
mm_lincomb(
  fit,
  weights,
  level = 0.95,
  method = c("auto", "satterthwaite", "kenward_roger", "asymptotic"),
  ...
)
```

Arguments

`fit`	A fitted `mm_lmm` or `mm_glmm`.
`weights`	A named numeric vector (or named list / single-row data.frame coercible to one). Names must match `names(fixef(fit))` exactly.
`level`	Confidence level for the Wald interval. Default 0.95.
`method`	For `mm_lmm`, the degrees-of-freedom method passed to `df_for_contrast()`. Defaults to `"auto"` (Satterthwaite when available). For `mm_glmm`, only `"asymptotic"` is accepted.
`...`	Reserved for future methods.

Details

For mm_glmm, the statistic is the asymptotic Wald z (no df). For mm_lmm, the default is Satterthwaite-approximated t via df_for_contrast(); pass method = "asymptotic" to force Wald z.

Weight names must be a subset of names(fixef(fit)). Coefficients not named in weights contribute zero. Pass the long-form contrast() front door if you need multiple contrasts or non-default rhs.

Value

A single-row data.frame with columns estimate, std_error, statistic, statistic_name ("t" or "z"), df, p_value, lower, upper, and method. The result carries an "mm_status" attribute reflecting the underlying vcov reliability (status, method, reliability, reason).

See Also

contrast() for the long-form, Rust-routed contrast surface with full estimability / reliability reporting.

Examples

```
## Not run:
# Difference-in-differences contrast at a focal SOA = 25 ms
# (Loo et al. 2026 aphantasia primary estimand, glmm path)
soa_s_25 <- (log(0.025) - mean(fit$data$soa_log)) / sd(fit$data$soa_log)
mm_lincomb(fit, c(
  "group: aphant:mask: masked"        = 1,
  "group: aphant:mask: masked:soa_s"  = soa_s_25
))

## End(Not run)
```

---

mm_lmm-methods *Extract components from a fitted mixeff LMM*

Description

These methods provide the common lme4-style extractor surface for lmm() fits. The required values are stored directly on the R object or rebuilt lazily from the serialized artifact, so these methods do not require a live Rust handle after saveRDS() / readRDS().

ngrps() returns a named integer vector giving the number of levels of each random-effect grouping factor, mirroring lme4::ngrps().

Produces the long form returned by as.data.frame(lme4::VarCorr(.)): one row per variance (var2 = NA) and one row per covariance (var1, var2 both set), with a final Residual row for LMMs. vcov holds the (co)variance and sdcor the standard deviation (diagonal) or correlation (off-diagonal). This is the shape broom.mixed::tidy() expects.

Produces the long form returned by `as.data.frame(lme4::ranef(.))`: columns `grpvar`, `term`, `grp`, `condval`, and `condsd`. `condsd` is the conditional standard deviation, taken from the `postVar` attribute when the modes were extracted with `condVar = TRUE`, and `NA` otherwise.

Usage

```
fixef(object, ...)

## S3 method for class 'mm_lmm'
fixef(object, ...)

## S3 method for class 'mm_glmm'
fixef(object, ...)

ranef(object, ...)

## S3 method for class 'mm_lmm'
ranef(object, condVar = FALSE, ...)

## S3 method for class 'mm_glmm'
ranef(object, condVar = FALSE, ...)

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

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

VarCorr(x, ...)

## S3 method for class 'mm_lmm'
VarCorr(x, ...)

## S3 method for class 'mm_glmm'
VarCorr(x, ...)

## S3 method for class 'mm_lmm'
sigma(object, ...)

## S3 method for class 'mm_glmm'
sigma(object, ...)

## S3 method for class 'mm_lmm'
logLik(object, REML = NULL, ...)

## S3 method for class 'mm_glmm'
logLik(object, REML = NULL, ...)

## S3 method for class 'mm_lmm'
```

```
deviance(object, REML = NULL, ...)

## S3 method for class 'mm_glmm'
deviance(object, REML = NULL, ...)

## S3 method for class 'mm_lmm'
AIC(object, ..., k = 2)

## S3 method for class 'mm_glmm'
AIC(object, ..., k = 2)

## S3 method for class 'mm_lmm'
BIC(object, ...)

## S3 method for class 'mm_glmm'
BIC(object, ...)

## S3 method for class 'mm_lmm'
nobs(object, ...)

## S3 method for class 'mm_glmm'
nobs(object, ...)

## S3 method for class 'mm_lmm'
df.residual(object, ...)

## S3 method for class 'mm_glmm'
df.residual(object, ...)

## S3 method for class 'mm_lmm'
formula(x, ...)

## S3 method for class 'mm_glmm'
formula(x, ...)

## S3 method for class 'mm_lmm'
model.frame(formula, ...)

## S3 method for class 'mm_glmm'
model.frame(formula, ...)

ngrps(object, ...)

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

## S3 method for class 'mm_lmm'
ngrps(object, ...)
```

```
## S3 method for class 'mm_glmm'
ngrps(object, ...)

## S3 method for class 'mm_lmm'
weights(object, ...)

## S3 method for class 'mm_glmm'
weights(object, ...)

## S3 method for class 'mm_lmm'
extractAIC(fit, scale, k = 2, ...)

## S3 method for class 'mm_glmm'
extractAIC(fit, scale, k = 2, ...)

## S3 method for class 'mm_lmm'
terms(x, ...)

## S3 method for class 'mm_glmm'
terms(x, ...)

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

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

## S3 method for class 'mm_lmm'
model.matrix(object, type = c("fixed", "random"), ...)

## S3 method for class 'mm_glmm'
model.matrix(object, type = c("fixed", "random"), ...)

## S3 method for class 'mm_lmm'
vcov(object, type = c("fixed", "theta"), correlation = FALSE, ...)

## S3 method for class 'mm_glmm'
vcov(object, type = c("fixed", "theta"), correlation = FALSE, ...)
```

Arguments

`object`, `x`, `formula`, `fit`	A fitted `mm_lmm` or `mm_glmm` object.
`...`	Reserved for generic compatibility.
`condVar`	Logical; when `TRUE`, Phase 2 returns the random-effects tables with an `NA` `postVar` array and an `mm_unavailable_reason` attribute rather than fabricating conditional variances.

REML	Ignored; included for S3 compatibility with likelihood and deviance generics.
k	Penalty per parameter for AIC().
scale	Ignored; included for S3 compatibility with extractAIC().
row.names, optional	Ignored; present for S3 consistency.
type	For model.matrix(), "fixed" returns the fixed-effect design matrix and "random" returns the sparse random-effect design matrix. For vcov(), "fixed" returns the fixed-effect covariance surface and "theta" returns an unavailable theta-covariance matrix with a reason attribute.
correlation	Logical; accepted for S3 compatibility with vcov().

Value

A named integer vector of group counts.

Examples

```
set.seed(1)
df <- data.frame(
  y = rnorm(60), x = rnorm(60),
  g = factor(rep(seq_len(10), each = 6))
)
fit <- lmm(y ~ x + (1 | g), df, control = mm_control(verbose = -1))
fixef(fit)
VarCorr(fit)
head(ranef(fit)$g)
sigma(fit)
logLik(fit)
nobs(fit)
```

---

mm_negative_binomial *Negative-binomial family for* glmm()

Description

NB2 family (variance mu + mu^2/theta, log link). With theta = NULL (the default) the size parameter is estimated alongside the model, matching lme4::glmer.nb(). Supplying a positive theta fits conditional on that value, matching glmer(family = MASS::negative.binomial(theta)) — which glmm() also accepts directly.

Usage

```
mm_negative_binomial(theta = NULL)
```

Arguments

theta	Optional positive NB2 size (dispersion) parameter. NULL estimates theta from the data.

Value

A family object accepted by glmm().

Examples

```
fam <- mm_negative_binomial()      # glmer.nb-style: theta estimated
fam_fixed <- mm_negative_binomial(theta = 2.5)
```

---

mm_parse_formula *Parse and canonicalize an lme4-style formula*

Description

mm_parse_formula() parses a formula string through the Rust formula parser and returns its canonical Display rendering. This is the Phase 0 round-trip primitive: equivalent formula spellings produce identical canonical strings, so equivalence-class testing in R is just string comparison on the canonical form.

Usage

```
mm_parse_formula(formula)
```

Arguments

formula	A single character string (length 1, non-NA, non-empty), or a one-sided / two-sided R formula object. R formula objects are coerced to character via format() before parsing.

Value

A single character string: the canonical rendering of the parsed formula.

Errors

Parse failures are signalled as a typed mm_formula_error condition (also inheriting from mm_condition and error). The condition object carries the original input string in its formula field. Catch with tryCatch(..., mm_formula_error = handler).

Examples

```
mm_parse_formula("y ~ x + (1 | g)")
mm_parse_formula(y ~ x + (1 | g))
```

model_report *Produce reporting tables for a fitted mixeff model*

Description

model_report() assembles a structured, publication-oriented report from the Rust artifact fields carried by a fitted model plus R-owned provenance such as the call and session metadata. reporting_table() extracts one section as a data-frame-compatible object.

Usage

```
model_report(fit, sections = "all", ...)

## S3 method for class 'mm_fit'
model_report(fit, sections = "all", ...)

reporting_table(object, section = "all", view = c("compact", "audit"), ...)

## S3 method for class 'mm_fit'
reporting_table(object, section = "all", view = c("compact", "audit"), ...)

## S3 method for class 'mm_model_comparison'
reporting_table(
  object,
  section = "comparison_ledger",
  view = c("compact", "audit"),
  ...
)

## S3 method for class 'mm_drop1'
reporting_table(
  object,
  section = "comparison_ledger",
  view = c("compact", "audit"),
  ...
)

## S3 method for class 'mm_random_effect_test'
reporting_table(object, section = "all", view = c("compact", "audit"), ...)
```

Arguments

fit	A fitted mm_fit, usually from lmm().
sections	Character vector of report sections, or "all".
...	Reserved for future methods.

object	For reporting_table(): a fitted mm_fit, an mm_model_report, or a comparison/test object with a durable ledger.
section	One section name, or "all".
view	"compact" for reader-facing columns, or "audit" for the full provenance table with source, reason, details, and related audit columns.

Value

model_report() returns an mm_model_report. reporting_table() returns an mm_reporting_table object: $table holds the section's data frame (or $sections the named list when section = "all").

---

optimizer_certificate *Inspect the optimizer certificate*

Description

Inspect the optimizer certificate

Usage

```
optimizer_certificate(object, ...)

## S3 method for class 'mm_compiled'
optimizer_certificate(object, ...)
```

Arguments

object	A compiled mm_spec or fitted mm_fit.
...	Reserved for future methods.

Value

An mm_optimizer_certificate object containing the raw certificate and a compact table view.

parameterization *Inspect covariance parameterization*

Description

`parameterization()` exposes the fitted theta/Lambda mapping recorded in the compiler artifact. It is the R table view of the upstream theta-map and covariance-parameter trace records.

Usage

```
parameterization(object, ...)

## S3 method for class 'mm_compiled'
parameterization(object, ...)
```

Arguments

`object`	A compiled `mm_spec` or fitted `mm_fit`.
`...`	Reserved for future methods.

Value

An `mm_theta_map` object with a data-frame `table` and raw trace records.

parametric_bootstrap *Parametric bootstrap likelihood-ratio comparison*

Description

Runs the engine-certified parametric-bootstrap likelihood-ratio test between two nested ML-fitted LMMs through the Rust `mm_bootstrap_lrt_json` entry point. The smaller model (fewer estimated parameters) is the reduced model; the larger is the alternative. The returned object carries the engine's replicate accounting (successful and completed replicates, boundary count, Monte-Carlo standard error, seed) rather than a bare `mean()` p-value, so every reported number traces back to a versioned Rust payload.

Usage

```
parametric_bootstrap(null, alternative, nsim = 100L, seed = NULL, ...)
```

Arguments

null, alternative	Fitted mm_lmm objects. Order is irrelevant; the model with fewer parameters is treated as the reduced model.
nsim	Number of bootstrap replicates.
seed	Optional bootstrap seed.
...	Reserved for future methods.

Details

The engine refuses REML fits: refit with lmm(..., REML = FALSE) before calling. (compare(method = "bootstrap") refits REML to ML automatically.)

Value

An mm_parametric_bootstrap object.

---

predict.mm_glmm *Predict from a fitted mixeff GLMM*

Description

GLMM predictions are computed on the R side from the stored fixed effects (population, re.form = NA) or fixed effects plus conditional modes (re.form = NULL), then mapped through the family link. This mirrors lme4::predict.merMod for generalized models: type = "link" returns the linear predictor and type = "response" the mean.

Usage

```
## S3 method for class 'mm_glmm'
predict(
  object,
  newdata = NULL,
  re.form = NULL,
  allow.new.levels = FALSE,
  type = c("response", "link"),
  se.fit = FALSE,
  interval = c("none", "confidence", "prediction"),
  level = 0.95,
  ...
)
```

Arguments

object	A fitted mm_glmm object.
newdata	Optional new data. Must be a data.frame containing every variable referenced by the model's formula. Categorical levels must either match the training factor levels or trigger the allow.new.levels policy.
re.form	Random-effects conditioning, following lme4's basic convention. NULL returns conditional predictions; NA (or ~0) returns population-level (fixed-effect) predictions. Conditioning on a subset of grouping factors via a one-sided formula is not supported by the current Rust contract and raises mm_inference_unavailable.
allow.new.levels	When FALSE (default), unseen grouping levels in newdata raise mm_inference_unavailable through the Rust NewReLevels::Error policy. When TRUE, unseen levels are replaced by the population mean (zero random effect), matching lme4::predict(allow.new.levels = TRUE).
type	Prediction scale. Gaussian LMMs use the same values for "response" and "link".
se.fit	Logical; when TRUE, returns a list with fit and se.fit. For population predictions (re.form = NA) the standard error is the Wald SE of the fixed-effect linear predictor, sqrt(diag(X V X')). For conditional predictions (re.form = NULL) the SE comes from the engine prediction-variance payload, which adds the random-effect contribution (BLUP variance and the fixed/random covariance). Rows the engine cannot certify — e.g. unseen grouping levels under allow.new.levels = TRUE — return NA with the engine's reason in the mm_reason attribute. (lme4::predict.merMod offers no conditional SE at all.)
interval	Interval type: "confidence" for the fitted mean or "prediction" for a new observation (adds the residual variance). Population (re.form = NA) intervals are fit +/- z*se computed R-side; conditional (re.form = NULL) bounds come from the engine prediction-variance payload. Returns a matrix with fit/lwr/upr.
level	Confidence level for interval / se.fit intervals.
...	Reserved for generic compatibility.

Details

In-sample response predictions reuse the engine's certified fitted values.

Standard errors and confidence intervals: population (re.form = NA) SEs are the fixed-effect Wald SE mapped through the link by the delta method; conditional (re.form = NULL) SEs and confidence bounds come from the engine prediction-variance payload. The engine certifies these rows for method = "joint_laplace" fits and for default pirls_profiled fits whose post-fit profiled-optimum certificate is issued (per-row status "available"). Uncertified fits (e.g. singular fits, or fits whose certificate fails) keep status "degraded", and their conditional SEs and bounds are withheld as NA with the engine's reason in the mm_reason attribute — consistent with the package's "no fake certainty" contract.

Prediction (future-observation) intervals (interval = "prediction") are available for conditional, response-scale predictions: the engine returns quantiles of the plug-in predictive distribution (the family conditional distribution mixed over link-scale fitted-mean uncertainty), so bounds are integers for count families and support points for Bernoulli. They are refused with a typed condition

on the link scale (future observations are response-scale objects), for population-level requests, and for grouped binomial fits (the future trial count is not representable in `newdata`).

Value

A numeric vector, or a list with `fit` and `se.fit` when `se.fit = TRUE`.

`predict.mm_lmm` *Predict from a fitted mixeff LMM*

Description

Predictions follow the lme4 generic shape. In-sample predictions reuse the cached fitted/fixed values; new-data predictions are dispatched through the Rust `predict_new` contract.

Usage

```
## S3 method for class 'mm_lmm'
predict(
  object,
  newdata = NULL,
  re.form = NULL,
  allow.new.levels = FALSE,
  type = c("response", "link"),
  se.fit = FALSE,
  interval = c("none", "confidence", "prediction"),
  level = 0.95,
  ...
)

## S3 method for class 'mm_lmm'
fitted(object, ...)

## S3 method for class 'mm_lmm'
residuals(
  object,
  type = c("response", "pearson", "deviance", "working"),
  scaled = FALSE,
  ...
)

## S3 method for class 'mm_glmm'
fitted(object, ...)

## S3 method for class 'mm_glmm'
residuals(object, type = c("response"), ...)
```

Arguments

object	A fitted mm_lmm object.
newdata	Optional new data. Must be a data.frame containing every variable referenced by the model's formula. Categorical levels must either match the training factor levels or trigger the allow.new.levels policy.
re.form	Random-effects conditioning, following lme4's basic convention. NULL returns conditional predictions; NA (or ~0) returns population-level (fixed-effect) predictions. Conditioning on a subset of grouping factors via a one-sided formula is not supported by the current Rust contract and raises mm_inference_unavailable.
allow.new.levels	When FALSE (default), unseen grouping levels in newdata raise mm_inference_unavailable through the Rust NewReLevels::Error policy. When TRUE, unseen levels are replaced by the population mean (zero random effect), matching lme4::predict(allow.new.levels = TRUE).
type	Prediction scale. Gaussian LMMs use the same values for "response" and "link".
se.fit	Logical; when TRUE, returns a list with fit and se.fit. For population predictions (re.form = NA) the standard error is the Wald SE of the fixed-effect linear predictor, sqrt(diag(X V X')). For conditional predictions (re.form = NULL) the SE comes from the engine prediction-variance payload, which adds the random-effect contribution (BLUP variance and the fixed/random covariance). Rows the engine cannot certify — e.g. unseen grouping levels under allow.new.levels = TRUE — return NA with the engine's reason in the mm_reason attribute. (lme4::predict.merMod offers no conditional SE at all.)
interval	Interval type: "confidence" for the fitted mean or "prediction" for a new observation (adds the residual variance). Population (re.form = NA) intervals are fit +/- z*se computed R-side; conditional (re.form = NULL) bounds come from the engine prediction-variance payload. Returns a matrix with fit/lwr/upr.
level	Confidence level for interval / se.fit intervals.
...	Reserved for generic compatibility.
scaled	Logical; when TRUE, residuals are divided by the residual scale.

Value

A numeric vector, or a list with fit and se.fit when se.fit = TRUE.

---

profile.mm_lmm *Profile a fitted linear mixed model*

Description

Computes profile-likelihood intervals for the model's parameters via the engine's certified profile payload and returns them as an mm_profile object: $table has one row per profiled parameter (parameter, estimate, lower, upper, regularity, reason_code). Under REML, fixed-effect coefficients are not profiled (upstream contract); their rows carry reason_code = "profile_beta_unavailable_under_reml" rather than being silently dropped. Use confint() with method = "profile" for the matrix form.

Usage

```
## S3 method for class 'mm_lmm'
profile(fitted, which = NULL, level = 0.95, ...)
```

Arguments

fitted	A fitted mm_lmm.
which	Optional character vector of parameter names to keep (coefficient names, "sigma", "theta1", ...).
level	Confidence level for the reported interval endpoints.
...	Unused; for generic consistency.

Value

An mm_profile object with $table, $level, $fit_criterion, and $notes.

See Also

confint() with method = "profile" for the matrix form.

---

random_blocks *Inspect random-effect blocks*

Description

random_blocks() summarizes the random-effect block structure recorded in the compiler artifact: grouping factor, basis, covariance family, theta parameter count, level counts, and design-support status.

Usage

```
random_blocks(object, ...)

## S3 method for class 'mm_compiled'
random_blocks(object, ...)
```

Arguments

object	A compiled mm_spec or fitted mm_fit.
...	Reserved for future methods.

Value

An mm_random_blocks object with a data-frame table.

random_options *Inspect nearby random-effect spellings for one grouping factor*

Description

`random_options()` is an opt-in map over nearby random-effect structures for a grouping factor. It recompiles each displayed spelling through the same upstream audit path as `compile_model()`, so support facts and block meanings come from Rust-authored `RandomTermCard` records.

Usage

```
random_options(spec, group, slope = NULL)
```

Arguments

`spec`	An `mm_spec` from `compile_model()` or, in later phases, an `mm_fit`.
`group`	Grouping factor to inspect. May be supplied bare (`group = subject`) or as a string.
`slope`	Optional slope variable to use for nearby slope-bearing spellings. When omitted, the function uses the first current random slope for `group`, then any scope-note fixed effect for `group`, then the first non-intercept fixed effect.

Value

An object of class `mm_random_options` with an `options` data frame, the upstream candidate `cards`, and the candidate audit reports.

refit *Refit a mixeff LMM with a new response*

Description

`refit()` fits the same model formula to a new response by calling `lmm()` with the stored model frame and `REML` setting.

Usage

```
refit(object, newresp, ...)

## S3 method for class 'mm_lmm'
refit(object, newresp, ...)
```

Arguments

object	A fitted mm_lmm.
newresp	Numeric response for refit().
...	Reserved for future methods.

Value

A new mm_lmm.

---

reproducibility *Inspect reproducibility metadata*

Description

Inspect reproducibility metadata

Usage

```
reproducibility(object, ...)

## S3 method for class 'mm_compiled'
reproducibility(object, ...)
```

Arguments

object	A compiled mm_spec or fitted mm_fit.
...	Reserved for future methods.

Value

An mm_reproducibility object.

---

revive *Revive a serialized mixeff object*

Description

revive() restores the process-local parts of a mixeff object after saveRDS() / readRDS() or a worker restart. The fitted artifact and flat extractor values are the durable source of truth; the Rust handle is only a cache and may be absent. In the current bridge, revival recreates the lazy R-side cache and explicitly leaves rust_handle = NULL.

Usage

```
revive(fit, ...)

## S3 method for class 'mm_fit'
revive(fit, ...)
```

Arguments

`fit`	A fitted `mm_fit` object.
`...`	Reserved for future methods.

Value

A revived `mm_fit` object.

`roles` *Declare or inspect design roles*

Description

`roles()` has two Phase 1.F uses. With named string arguments it constructs a declared-role object, e.g. `roles(subject = "sampled_unit")`. With a compiled spec or fit as its only unnamed argument, it returns the observed role fallback inferred from the formula and model frame.

Usage

```
roles(...)
```

Arguments

`...`	Either named role strings or one unnamed `mm_spec` / `mm_fit`.

Value

An `mm_roles` object with a data-frame `table`.

simulate.mm_lmm *Simulate from a mixeff LMM*

Description

Draws Gaussian responses from the stored fixed effects, random-effect covariance summaries, and residual scale.

Usage

```
## S3 method for class 'mm_lmm'
simulate(object, nsim = 1, seed = NULL, re.form = NULL, ...)
```

Arguments

object	A fitted mm_lmm.
nsim	Number of simulated responses.
seed	Optional random seed.
re.form	Random-effects conditioning. NULL simulates new random effects; NA simulates from the population-level mean only.
...	Reserved for future methods.

Value

A data frame of simulated responses.

test_effect *Test a fixed-effect term*

Description

test_effect() asks Rust to construct fixed-effect term hypotheses and returns the corresponding fixed-effect inference rows.

Usage

```
test_effect(
  fit,
  term,
 method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "bootstrap_lrt",
    "cluster_bootstrap", "asymptotic", "boundary_lrt", "none"),
  bootstrap = NULL,
  group = NULL,
  ...
```

```
)

## S3 method for class 'mm_lmm'
test_effect(
  fit,
  term,
 method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "bootstrap_lrt",
    "cluster_bootstrap", "asymptotic", "boundary_lrt", "none"),
  bootstrap = NULL,
  group = NULL,
  ...
)
```

Arguments

`fit`	A fitted `mm_lmm`.
`term`	A fixed-effect term label.
`method`	Requested inference method.
`bootstrap`	Optional `bootstrap_control()` object for bootstrap-backed methods.
`group`	Optional grouping factor for `method = "cluster_bootstrap"`. Required for crossed or multi-grouping-factor models. In schema 1.0.0, cluster resampling is an estimator-distribution target and term-level p-values return `not_assessed` with a stable reason code.
`...`	Reserved for future methods.

Value

An `mm_effect_test` object.

`test_random_effect` *Test a random-effect variance component*

Description

`test_random_effect()` exposes the boundary-aware likelihood-ratio route for random-effect variance components. The v1 certified route is a nested ML comparison that adds exactly one variance/covariance parameter and reports the Self-Liang 50:50 mixture reference distribution. It is intentionally separate from `test_effect()`, which tests fixed effects.

Usage

```
test_random_effect(
  fit,
  term,
  method = c("boundary_lrt"),
  refit_for_comparison = c("auto", "error", "ml"),
```

```
  ...
)

## S3 method for class 'mm_lmm'
test_random_effect(
  fit,
  term,
  method = c("boundary_lrt"),
  refit_for_comparison = c("auto", "error", "ml"),
  ...
)
```

Arguments

fit	A fitted mm_lmm.
term	Random-effect term to test. This can be the term id ("r0"), the original random-effect fragment such as "(1 | subject)", or a unique grouping factor name such as "subject".
method	Currently "boundary_lrt".
refit_for_comparison	How to handle REML fits. "auto" and "ml" refit to ML; "error" refuses.
...	Reserved for future methods.

Value

An mm_random_effect_test object with a one-row table.

---

update.mm *Update and re-fit a mixeff model*

Description

update() re-fits an mm_lmm or mm_glmm with a modified model specification, mirroring stats::update() and lme4's update() for the common cases: changing the formula (. ~ . - x), toggling REML, swapping weights/offset/family/control, or supplying new data.

Usage

```
## S3 method for class 'mm_lmm'
update(object, formula., ..., evaluate = TRUE)

## S3 method for class 'mm_glmm'
update(object, formula., ..., evaluate = TRUE)
```

Arguments

`object`	A fitted `mm_lmm` or `mm_glmm`.
`formula.`	A formula-change applied with `stats::update.formula()`; omit to keep the current formula. Random-effect terms (`(x | g)`, `(x || g)`) are preserved across `. ~ .` edits.
`...`	Arguments to override on the re-fit. For `mm_lmm`: `data`, `REML`, `weights`, `control`. For `mm_glmm`: additionally `family`, `offset`, `method`, `nAGQ`, `inference`.
`evaluate`	If `TRUE` (default) re-fit and return the new model; if `FALSE` return the unevaluated call.

Details

The re-fit reuses the fitted model frame (`model.frame()`) as the default data source, so formula edits that *remove* terms or change estimation options work without re-supplying data. A formula edit that introduces a **new** variable absent from the original model frame requires an explicit `data =` argument.

Value

A new fitted model of the same class as `object`, or an unevaluated call when `evaluate = FALSE`.

Examples

```
set.seed(1)
df <- data.frame(
  y = rnorm(80), x = rnorm(80), z = rnorm(80),
  g = factor(rep(seq_len(10), each = 8))
)
fit <- lmm(y ~ x + z + (1 | g), df, control = mm_control(verbose = -1))
# drop a fixed term
fit2 <- update(fit, . ~ . - z)
# refit by ML for a likelihood-ratio comparison
fit_ml <- update(fit, REML = FALSE)
fixef(fit2)
```

`verify_convergence` *Verify convergence of a fitted linear mixed model*

Description

`verify_convergence()` re-runs the fit under the engine's bounded verification workflow and reports whether the extra runs agree with the fitted optimum: a restart from the optimum, one or more jittered restarts, and (opt-in) an alternate-optimizer consensus pass. Agreement is judged by the engine against the objective/theta/beta tolerances below; the verdict (`status`), the per-run deltas, and the wording are all owned by the Rust contract — R only formats them.

Usage

```
verify_convergence(fit, ...)

## Default S3 method:
verify_convergence(fit, ...)

## S3 method for class 'mm_lmm'
verify_convergence(
  fit,
  ...,
  restart = TRUE,
  jitter_starts = 1L,
  jitter_scale = 1e-04,
  consensus = FALSE,
  max_feval = 500L,
  objective_tolerance = 1e-05,
  theta_tolerance = 0.001,
  beta_tolerance = 1e-04
)
```

Arguments

`fit`	A fitted `mm_lmm` from `lmm()`.
`...`	Reserved for future methods.
`restart`	Logical; re-optimize starting from the fitted optimum and compare against it.
`jitter_starts`	Number of restarts from jittered copies of the fitted covariance parameters.
`jitter_scale`	Relative scale of the jitter applied to theta.
`consensus`	Logical; also refit with an engine-chosen alternate optimizer and compare. Default `FALSE`: this vendored build compiles without the optional `nlopt` backend, and for some models the engine's alternate choice is an nlopt optimizer — its absence would then be reported as a non-agreeing run (status `fragile`) that reflects the build, not the fit. Enable it when you want the consensus pass and will read the per-run diagnostics.
`max_feval`	Positive integer cap on objective evaluations per verification run.
`objective_tolerance, theta_tolerance, beta_tolerance`	Positive agreement tolerances on the objective value, the covariance parameters, and the fixed effects.

Details

The verifier refits the model from the stored specification before it starts, so a call costs roughly `2 + jitter_starts` fits (plus consensus runs when enabled).

Value

An object of class `mm_convergence_verification` carrying:

`status` the engine verdict: `not_run`, `restart_agrees`, `optimizer_consensus`, `fragile`, or `unstable`

`message` the engine's one-line summary

`table` a data frame with one row per verification run (label, optimizer, return code, objective/theta/beta deltas, agreement)

`reference` the reference optimum the runs were compared to

`tolerances` the agreement tolerances that were applied

`raw` the parsed engine payload

See Also

`optimizer_certificate()` for what the original fit ran; `mm_control()` to refit with a different optimizer or tolerances.

Examples

```
## Not run:
fit <- lmm(y ~ t + (1 | s), df)
verify_convergence(fit)

## End(Not run)
```

Index