

Package: mixeff (via r-universe)

October 15, 2026

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

Version 0.2.0

Copyright file inst/COPYRIGHTS

Description Fits linear and generalized linear mixed-effects models from 'lme4'-style formulas, with optimization by the bundled 'mixeff-rs' Rust crate. A fitted model stores the compiled model specification, the optimizer certificate, and per-coefficient inference metadata (method, status, reliability) as data, so a result can be inspected, saved, and reloaded without a live handle to the compiled engine. When an inference method is unavailable for a fit, the affected value is either withheld with a stable reason code or reported under the name of the method actually used, never silently swapped. Vignettes cover migration from 'lme4' and the meaning of random-effects formula 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

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager, >= 1.85.0), rustc (>= 1.85.0), GNU make

Depends R (>= 4.2)

Imports graphics, grDevices, jsonlite, Matrix, methods, rlang, stats, utils

Suggests boot, broom, broom.mixed, generics, insight, ggplot2, emmeans
 (>= 1.4), estimability, jsonvalidate, knitr, lattice, lme4,
 lmerTest, MASS, nlme, parallel, pbkrtest, performance, pkgload,
 processx, reformulas, rextendr, rmarkdown, statmod, testthat
 (>= 3.0.0)

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

Config/Needs/website albersdown

Config/pak/sysreqs make libclang-dev

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

Date/Publication 2026-10-10 15:54:38 UTC

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

RemoteRef HEAD

RemoteSha 6e9a62a53f1b06e76ae56c490f1b221aa5a6a786

Contents

anova.mm_glmm	4
anova.mm_lmm	4
as_json	6
audit	6
audit_design	8
bootstrap_control	8
changes	9
compare	10
compare_covariance	11
compile_model	12
confint.mm_glmm	13
confint.mm_lmm	14
contrast.mm_glmm	16
df_for_contrast	17
diagnostics	18
drop1.mm_glmm	19
drop1.mm_lmm	19
estimability	20
explain_model	21
fit_handle_alive	22
getME	22
glmm	24
inference_options	26
inference_table	27
is_singular	28
lmm	29
mm_allfit	31
mm_bootmer	32

mm_control	33
mm_family	35
mm_formula_manifest	36
mm_grid	37
mm_influence	40
mm_json_known_schemas	42
mm_json_negotiate	43
mm_lincomb	44
mm_lmlist	45
mm_lmm-methods	47
mm_negative_binomial	52
mm_parse_formula	52
mm_variance_components	53
model_report	55
optimizer_certificate	56
parameterization	57
parametric_bootstrap	57
plot.mm_fit	58
plot.mm_ranef	60
predict.mm_glmm	61
predict.mm_lmm	63
profile.mm_lmm	65
random_blocks	66
random_options	67
refit	68
REMLcrit	69
rePCA	70
reproducibility	71
revive	71
roles	72
simulate.mm_lmm	72
step	74
step.mm_lmm	75
supported_models	76
test_effect	77
test_random_effect	78
update.mm	79
verify_convergence	80

anova.mm_glmm	<i>Analysis of deviance for GLMMs</i>
---------------	---------------------------------------

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

A data frame of class `c("mm_anova_comparison", "mm_glmm_comparison", "anova", "data.frame")` shaped like `anova(glmer1, glmer2)`: columns `npar`, `AIC`, `BIC`, `logLik`, `-2*log(L)` (lme4 before 2.0 called it deviance), `Chisq`, `Df`, `Pr(>Chisq)`, rows named after the arguments. The underlying comparison (with the model formulas) is reachable as `x$table`.

anova.mm_lmm	<i>Analysis of variance for a mixeff LMM</i>
--------------	--

Description

With no extra models, `anova()` returns a term-level F-test table for object shaped like `lmerTest`'s: a data frame of class `"anova"` with columns `Sum Sq`, `Mean Sq`, `NumDF`, `DenDF`, `F value` and `Pr(>F)` (rows named by term; $\text{Mean Sq} = F * \sigma^2$ and $\text{Sum Sq} = \text{NumDF} * \text{Mean Sq}$, as in `lmerTest`). With further fitted models in `...`, it returns lme4's likelihood-ratio table (`npar`, `AIC`, `BIC`, `logLik`, `-2*log(L)`, `Chisq`, `Df`, `Pr(>Chisq)`), rows named after the arguments; lme4 before 2.0 called the `-2*log(L)` column deviance), computed by `compare()` (REML fits are refit by ML, as in lme4).

Usage

```
## S3 method for class 'mm_lmm'
anova(
  object,
  ...,
  type = c("III", "II", "I", "block"),
  method = c("auto", "satterthwaite", "kenward_roger", "bootstrap", "asymptotic", "none"),
  refit_for_comparison = c("auto", "error", "ml"),
  ddf = NULL
)
```

Arguments

<code>object</code>	A fitted <code>mm_lmm</code> .
<code>...</code>	Optional additional fitted models; triggers <code>compare()</code> .
<code>type</code>	Term-hypothesis type. "III" (default) tests marginal Type III hypotheses: a term's own contrast columns plus the equally-weighted average over the levels of every term that contains it, which makes the test invariant to which factor level is the reference and matches SAS / car / lmerTest Type III. "II" respects marginality (each term adjusted for all terms that do not contain it). "I" is sequential in <code>terms()</code> order (main effects, then two-way interactions, and so on). "block" tests the raw coefficient block for each term — under treatment coding that is the simple effect at the other factors' reference levels, which is a legitimate quantity but is <i>not</i> Type III on unbalanced designs; it is the hypothesis <code>mixeff</code> computed for "III" before engine 1f3f689. lmerTest's spellings 3, 2, 1 (numeric or character) are accepted.
<code>method</code>	Degrees-of-freedom / test method.
<code>refit_for_comparison</code>	Passed to <code>compare()</code> when <code>...</code> is used.
<code>ddf</code>	lmerTest's spelling of the method: "Satterthwaite" (<code>method = "satterthwaite"</code>), "Kenward-Roger" (<code>method = "kenward_roger"</code>), or "lme4", which returns lme4's own single-model table (sequential <code>npair</code> , Sum Sq, Mean Sq, F value, no denominator df or p-values). Supply <code>ddf</code> or <code>method</code> , not both.

Details

`mixeff`'s provenance stays on the result: `x$table` is the full term-level (or `compare()`) table with `method`, `status`, `reliability` and `reason` columns, and `x$type` / `x$requested_method` record the request. Rows the engine could not certify keep NA statistics and their reason is printed under the table.

Value

A data frame of class `c("mm_anova", "anova", "data.frame")` for one model, or `c("mm_anova_comparison", "mm_model_comparison", "anova", "data.frame")` for several.

as_json	<i>Serialize a mixeff spec or fit to JSON</i>
---------	---

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	<i>Audit a compiled model spec or fitted model</i>
-------	--

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 uncompileable 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

```
set.seed(1)
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))
```

audit_design	<i>Deprecated alias for audit()</i>
--------------	---

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	<i>Fixed-effect bootstrap control</i>
-------------------	---------------------------------------

Description

Fixed-effect bootstrap control

Usage

```
bootstrap_control(
  nsim = 999L,
  seed = NULL,
  failed_refit_policy = c("exclude", "count_extreme", "abort"),
  threads = 1L
)
```

Arguments

<code>nsim</code>	Requested bootstrap replicate count.
<code>seed</code>	Optional integer seed. NULL (the default) draws a seed from R's random number generator when the bootstrap runs, so <code>set.seed()</code> makes the bootstrap reproducible; the seed used is recorded in the result.
<code>failed_refit_policy</code>	How failed refits are accounted for. Stable Rust wire labels are "exclude", "count_extreme", and "abort".
<code>threads</code>	Number of worker threads for the replicate refits (default 1, serial). See the "Threads" section.

Value

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

Threads

The engine simulates every replicate's response serially, in the serial random-number order, and refits the replicates on threads worker threads. The result is identical for every threads value; only the wall-clock time changes. Workers never call into R: the R thread alone polls for interrupts, so Ctrl-C / Esc still stops the run. The default is 1 because CRAN policy limits packages to at most two threads unless the user explicitly asks for more; passing threads is that request (for example `threads = parallel::detectCores()`).

Examples

```
bootstrap_control(nsim = 199, seed = 1, threads = 2)
```

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	<i>Compare fitted mixeff models</i>
---------	-------------------------------------

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", "kenward_roger", "satterthwaite"),
  refit_for_comparison = c("auto", "error", "ml"),
  nsim = 0L,
  seed = NULL,
  threads = 1L
)
```

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, "bootstrap" for a small parametric-bootstrap LRT when nsim > 0, or "kenward_roger" / "satterthwaite" for pbkrtest's KRmodcomp() / SATmodcomp(): an F test of the larger model's fixed effects restricted to the smaller model's (two models with identical random effects, nested in their fixed effects). The restriction matrix is built from the two design matrices as pbkrtest does; the test runs on the REML fit of the larger model (refitted by REML if it was fitted by ML, as pbkrtest does). The Kenward-Roger F is pbkrtest's scaled

statistic (the main Ftest row of KRmodcomp(), which lmerTest's anova(ddf = "Kenward-Roger") also reports): $F = \lambda F_U$ referred to $F(q, \nu)$ with the KR denominator df, where λ is the KR scaling factor (f_scaling; 1 for a single restriction). The unscaled statistic and its p-value (pbkrtest's FtestU row) are kept as unscaled_statistic / unscaled_p_value. The F row replaces the LRT row (statistic_name = "F", df = numerator df, den_df = denominator df); the full result is in \$fixed_f.

refit_for_comparison	How to handle REML fits.
nsim	Number of bootstrap simulations for method = "bootstrap".
seed	Optional bootstrap seed.
threads	Worker threads for the bootstrap refits (default 1; see bootstrap_control()). The result is identical for every value.

Value

An mm_model_comparison object with a data-frame table.

compare_covariance	<i>Compare covariance parameterizations for current random terms</i>
--------------------	--

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 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 `lmm()` all consume the same artifact.

Usage

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

Arguments

<code>formula</code>	A two-sided lme4-style formula, e.g. <code>y ~ x + (1 + x subject)</code> .
<code>data</code>	A <code>data.frame</code> whose columns include every variable named in <code>formula</code> . Variables with missing values raise an <code>mm_data_error</code> ; pass <code>na.omit(data)</code> 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).

Compiling 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; `lmm()` and `glmm()` fit the model itself.

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

```
set.seed(1)
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)
```

Description

Two routes are available. `method = "asymptotic"` builds Wald intervals (estimate $\pm z \cdot \text{SE}$) from the Rust fixed-effect inference table; it requires a certified fit (`method = "joint_laplace"`) or the explicit working-Hessian opt-in, and refuses the profiled (`pirls_profiled`) estimator with a typed reason. `method = "bootstrap"` runs a parametric bootstrap — each replicate simulates a response from the fitted model under fresh random-effect draws and refits the same (effective) estimator — and returns percentile intervals; it works on profiled-estimator fits for every supported family, including negative binomial (fixed-theta fits condition replicates on that theta; estimated-theta fits re-estimate theta per replicate, propagating its uncertainty). On `joint_laplace` fits the bootstrap is refused at this engine pin — replicate refits cannot re-run the joint estimator — and those fits have certified Wald intervals instead. The bootstrap result carries replicate accounting as attributes: `mm_bootstrap` (requested, successful, failed, seed, per-parameter standard errors, Monte Carlo SE, and a reliability grade) and `mm_estimator`. Failed replicates are counted, never silently dropped. Profile intervals are not certified for GLMMs and are refused.

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), its synonyms "wald" and lme4's "Wald", or "bootstrap" (lme4's spelling "boot" is accepted). lme4 defaults to "profile"; mixeff keeps the Wald default because profile intervals are refused for GLMMs. The same spellings are accepted by the LMM method (confint.mm_lmm()), whose "profile" intervals use lme4's .sig01, ..., .sigma rows.
...	For method = "bootstrap": nsim (number of replicates, default 999), seed (non-negative integer; when omitted one is drawn from R's RNG so set .seed() governs reproducibility), and threads (worker threads for the replicate refits, default 1; the result is identical for every value – see bootstrap_control() for the threading contract and CRAN's two-thread default policy).

Value

An mm_confint matrix of lower/upper bounds ("asymptotic"), or a plain matrix of percentile bounds with bootstrap-accounting attributes ("bootstrap").

confint.mm_lmm	<i>Confidence intervals for a mixeff LMM</i>
----------------	--

Description

method = "asymptotic" (the default; synonyms "wald" and lme4's "Wald") gives Wald intervals for the fixed effects from the stored standard errors. method = "profile" gives profile-likelihood intervals with lme4's rows and names: .sig01, .sig02, ... (random-effect standard deviations and correlations, numbered as lme4 >= 2.0 does: term by term, each term's standard deviations before its correlations), .sigma, then the fixed effects. As in lme4, every fit is profiled on the ML deviance: a REML fit's intervals are those of its ML refit. The engine's relative-Cholesky theta1, ... intervals (on the fit's own criterion) are kept in attr(ci, "mm_profile")\$table and can be requested by name through parm. When one parameter's profile is irregular (non-monotone, not

bracketing the estimate, or a failed refit), its undetermined bounds are NA – lme4 would warn and interpolate linearly, often reporting [-1, 1] for a correlation – and the reason is in `attr(ci, "mm_profile")$table (status, reason_code, reason)`; printing the result adds a note. See `profile.mm_lmm(). method = "bootstrap"` (lme4's "boot") gives parametric-bootstrap intervals for the fixed effects, controlled by `bootstrap_control()`.

Usage

```
## S3 method for class 'mm_lmm'
confint(
  object,
  parm,
  level = 0.95,
  method = c("asymptotic", "wald", "bootstrap", "profile"),
  bootstrap = NULL,
  interval = c("percentile", "basic"),
  threads = 1L,
  ...
)
```

Arguments

<code>object</code>	A fitted <code>mm_lmm</code> .
<code>parm</code>	Parameters to report: names (fixed-effect names, and for "profile" also ".sig01", ..., ".sigma", "theta1", ...), lme4's "theta_" / "beta_" shortcuts ("profile" only), or indices. Defaults to all fixed effects (all lme4 rows for "profile").
<code>level</code>	Confidence level.
<code>method</code>	"asymptotic", "wald", "bootstrap", or "profile".
<code>bootstrap</code>	Optional <code>bootstrap_control()</code> for <code>method = "bootstrap"</code> (its <code>threads</code> field sets the bootstrap's worker threads).
<code>interval</code>	Bootstrap interval type, "percentile" or "basic".
<code>threads</code>	Worker threads for <code>method = "profile"</code> (default 1). The per-parameter profiles run concurrently; the result is identical for every value. The default is 1 because CRAN policy limits packages to at most two threads unless the user explicitly asks for more; passing threads is that request.
<code>...</code>	Unused.

Details

lme4's `confint.merMod()` defaults to `method = "profile"`; `mixeff` keeps the Wald default because the profile refits the model many times (about the cost of lme4's profile; several seconds for a modest model) and can refuse on near-singular fits, while the Wald route is free. Request `method = "profile"` to reproduce lme4's default output.

Value

An `mm_confint` matrix of lower/upper bounds.

Examples

```
set.seed(1)
d <- data.frame(g = factor(rep(1:10, each = 6)), x = rnorm(60))
d$y <- 1 + 0.5 * d$x + rnorm(10)[d$g] + rnorm(60)
fit <- lmm(y ~ x + (1 | g), d, control = mm_control(verbose = -1))
confint(fit)
confint(fit, method = "profile")
```

contrast.mm_glmm	<i>Contrast fixed effects</i>
------------------	-------------------------------

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 <code>bootstrap_control()</code> 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, $L \beta_{\text{hat}} - \text{rhs}$.

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

<code>fit</code>	A fitted <code>mm_lmm</code> .
<code>L</code>	A contrast vector or matrix.
<code>method</code>	Requested degrees-of-freedom method.
<code>...</code>	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.

<code>diagnostics</code>	<i>Inspect mixeff diagnostics and fit status</i>
--------------------------	--

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

<code>fit</code>	A compiled <code>mm_spec</code> or fitted <code>mm_fit</code> .
<code>severity</code>	Optional character vector used to filter diagnostics by severity.
<code>stage</code>	Optional character vector used to filter diagnostics by stage.
<code>...</code>	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	<i>Drop one fixed-effect term at a time from a GLMM</i>
---------------	---

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(glmMod, test = "Chisq")`.

Usage

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

Arguments

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

Value

An `mm_drop1` object.

drop1.mm_lmm	<i>Drop one fixed-effect term at a time</i>
--------------	---

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

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

Value

An `mm_drop1` object.

<code>estimability</code>	<i>Assess contrast estimability</i>
---------------------------	-------------------------------------

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

<code>fit</code>	A fitted <code>mm_lmm</code> .
<code>L</code>	Optional contrast vector or matrix. Defaults to the fixed-effect coefficient basis.
<code>...</code>	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 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

```
set.seed(1)
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))
```

fit_handle_alive	<i>Test whether a mixeff fit has a live native handle</i>
------------------	---

Description

The native handle is a process-local cache of the fitted engine model, created by `lmm()` and `glmm()`. While it is alive, computations that need the engine model (contrasts, `summary()` tests, predictions with intervals, `ranef(condVar = TRUE)`, profiles, bootstraps, model comparison, `verify_convergence()`) reuse it instead of refitting. A FALSE result does not mean the fit is unusable: the handle does not survive `saveRDS()` / `readRDS()` or a new R process, and those computations then refit the model from the stored model frame, with identical results; extractors read from the durable artifact and flat R-side payload, and `revive()` recreates the lazy cache after serialization. Set `options(mixeff.keep_handle = FALSE)` before fitting to not keep handles (each one holds a copy of the model data for as long as the fit object lives).

Usage

```
fit_handle_alive(fit, ...)

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

Arguments

<code>fit</code>	A fitted <code>mm_fit</code> object.
<code>...</code>	Reserved for future methods.

Value

A length-one logical value.

getME	<i>Extract low-level model components</i>
-------	---

Description

`getME()` mirrors `lme4::getME()` for `lmm()` and `glmm()` fits. Every component is rebuilt R-side from quantities the fit stores (fixed effects, theta and its map, conditional modes, fitted values, the model frame), so it works on fits restored with `readRDS()`.

Usage

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

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

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

Arguments

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

Details

Supported components: "X", "Z", "Zt", "Ztlist", "mmList", "y", "mu", "u", "b", "Gp", "Tp", "L", "Lambda", "Lambdat", "Lind", "Tlist", "ST", "A", "RX", "RZX", "sigma", "flist", "fixef", "beta", "theta", "REML", "is_REML", "n_rtrms", "n_rfacs", "N", "n", "p", "q", "p_i", "l_i", "q_i", "k", "m_i", "m", "cnms", "devcomp", "offset", "weights", "lower", "glmer.nb.theta" (negative-binomial GLMMs), and "ALL" (a named list of every available component).

"L" is the sparse Cholesky factor (a Matrix CHMfactor, with the same fill-reducing permutation lme4 uses) of $\text{Lambda}'Z'WZ \text{ Lambda} + I$, where W holds the prior weights for an LMM and the final PIRLS working weights for a GLMM. "RZX", "RX", "u" and "devcomp" (cmp and dims) are computed from the same decomposition at the fitted parameters. "u" is the minimum-norm spherical mode with $b = \text{Lambda } u$, which is the penalized optimum also when Lambda is singular.

"devfun" is refused: the optimizer runs inside the Rust engine and no R deviance function exists. Refusals, and requests for unknown components, raise an mm_inference_unavailable condition with reason_code = "getme_component_unavailable" naming the component.

Value

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

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))
getME(fit, "theta")
getME(fit, "lower")
getME(fit, "devcomp")$cmp
```

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 = "joint_laplace"` is the labeled joint Laplace route (`fast = FALSE`, `nAGQ = 1`), the estimator `lme4::glmer()` uses by default, backed by the native dependency-light optimizer in this vendored build. `method = "pirls_profiled"` is the labeled fast-PIRLS profiled path (`lme4`'s `nAGQ = 0`).

Usage

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

Arguments

<code>formula</code>	A two-sided <code>lme4</code> -style formula.
<code>data</code>	A <code>data.frame</code> .
<code>family</code>	A supported GLMM family object or family constructor. The supported surface (every family/link pair the engine fits) is: <code>binomial()</code> with "logit", "probit", or "cloglog" links; <code>poisson()</code> with "log" or "sqrt" links; <code>Gamma()</code> with "inverse" (R's default) or "log" links; <code>inverse.gaussian()</code> with "inverse" or "log" links (R's default "1/mu^2" link is not available and is refused); <code>gaussian()</code> with non-identity "log", "inverse", or "sqrt" links (a Gaussian identity-link model is an LMM: use <code>lmm()</code>); and negative binomial (NB2, "log" link) via <code>mm_negative_binomial()</code> (theta estimated, like <code>lme4::glmer.nb()</code>) or <code>MASS::negative.binomial(theta)</code> (fixed theta). Binomial "cauchit", "log", and "identity" and Poisson "identity" links are not available in the engine and are refused with a typed <code>mm_inference_unavailable</code> error naming the supported set.

random	Reserved for the native random-effect constructor path.
weights	Optional prior weights: a column of data (as in lme4, weights = trials) or a numeric vector. For binomial models these are trial counts for proportion responses; weights must be positive and finite.
offset	Optional fixed linear-predictor offset: a column of data or a numeric vector; values must be finite. offset() terms in the formula are also supported and are added to it (as in stats::glm()).
subset, na.action, contrasts	As in lmm(): subset selects rows (evaluated in data), na.action controls missing values (default getOption("na.action"), i.e. na.omit, as in glmer()); only model variables, weights and offset count; dropped rows are announced with a typed mm_rows_dropped message and recorded in na.action(fit); na.exclude pads fitted()/residuals()/predict() back to the original rows; na.fail and na.pass are refused with a typed mm_data_error, and contrasts is honoured only when it names the engine's coding (contr.treatment for unordered, contr.poly for ordered factors).
method	GLMM estimation method. "joint_laplace" (the default) is the joint Laplace route, matching glmer()'s default estimator, and requires nAGQ = 1; it certifies Wald standard errors, tests, and intervals. Its optimizer cost is higher than the profiled path's; cap it with mm_control(max_feval =). "pirls_profiled" is the fast profiled PIRLS path (equivalent to lme4's nAGQ = 0 fast estimate); its coefficients do not match glmer(nAGQ = 1) exactly and Wald inference is withheld. When method is not supplied and the request needs the profiled path – a negative-binomial family (the joint route is not available for it yet), nAGQ > 1, or inference = "working_hessian" – glmm() uses "pirls_profiled" and says so with an mm_estimator_notice message (silence with mm_control(verbose = -1)). An explicit method = "joint_laplace" with a negative-binomial family or nAGQ != 1 is refused, never swapped.
nAGQ	Number of adaptive Gauss-Hermite quadrature points. 1 (the default) is the Laplace approximation. 0 requests lme4's PIRLS-only fast estimate and selects method = "pirls_profiled" (refused together with an explicit method = "joint_laplace"). Values above 1 run on the profiled path only and, as in lme4, require a model with a single scalar random-effect term (e.g. (1 g)); other models are refused.
inference	Requested inference posture. The default "auto" keeps the certified contract: Wald standard errors, tests, and intervals are available only when the engine certifies them (currently method = "joint_laplace", the default); the profiled estimator withholds them with a typed refusal. "working_hessian" is an explicit opt-in that unlocks the UNCERTIFIED profiled working-Hessian approximation on every inference route; each resulting row is labelled wald_z_working_hessian with reliability moderate. Its standard errors came within 1% of glmer()'s on the package's reference dataset, but nothing certifies that agreement in general, so treat it as an exploration and screening tool, not a reporting route; see inference_options(). "none", "asymptotic", and "bootstrap" are accepted and recorded but currently equivalent to "auto".
control	A list from mm_control(). Besides the optimizer settings it carries lme4 2.1-0's glmerControl() dispersion controls (disp_method, disp_dof_correction,

max_phi_iter for lme4's maxPhiIter), which affect Gamma, inverse-Gaussian and Gaussian non-identity-link fits. The control is stored on the fit and reused by refit(), update() and the internal refits (bootstrap, profiles, predictions).

... 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. The fit checks for a user interrupt (Ctrl-C / Esc) between optimizer evaluations and stops with a typed mm_interrupted error (the check happens at evaluation boundaries, so a single very expensive evaluation finishes first). Evaluation budgets are bounded (a bounded budget caps optimizer iterations; it does not prove every native evaluation terminates); runtime on large problems is governed by mm_control(max_feval =).

Value

An object of class mm_glm, 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))
)
# Default: glmer-equivalent joint Laplace estimates.
fit <- glmm(y ~ x + (1 | g), df, family = binomial(),
  control = mm_control(verbose = -1))
fixef(fit)
# Fast profiled PIRLS path (lme4's nAGQ = 0):
fit_fast <- glmm(y ~ x + (1 | g), df, family = binomial(),
  method = "pirls_profiled",
  control = mm_control(verbose = -1))
fixef(fit_fast)
```

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, ...)

## S3 method for class 'mm_glmm'
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	<i>Fixed-effect inference table</i>
-----------------	-------------------------------------

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"),
  ...
)
```

```
)

## S3 method for class 'mm_glm'
inference_table(fit, method = "auto", ...)
```

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	<i>Test whether a fit is singular or reduced-rank</i>
-------------	---

Description

Follows `lme4::isSingular()`: a fit is singular when any covariance parameter with a lower bound of zero (a diagonal element of a relative covariance factor, `getME(x, "lower") == 0`) is below `tol`. Works for `lmm()` and `glmm()` fits. `lme4::isSingular()` is a plain function, so it cannot dispatch on `mixeff` fits; call `is_singular()` instead. Fits without a stored theta map (very old serialized objects) fall back to the engine's fit-status and effective-covariance labels.

Usage

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

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

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

Arguments

x	A fitted mm_lmm or mm_glm.
tol	Tolerance on the zero-bounded theta values (default 1e-4, as in lme4).
...	Reserved for future methods.

Value

A length-one logical value.

lmm	<i>Fit a linear mixed-effects model</i>
-----	---

Description

`lmm()` is `mixeff`'s 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 = getOption("na.action"),
  contrasts = NULL,
  control = mm_control(),
  offset = NULL
)
```

Arguments

formula	A two-sided <code>lme4</code> -style formula, e.g. <code>y ~ x + (1 + x subject)</code> .
data	A <code>data.frame</code> containing all variables in <code>formula</code> .
REML	Logical; fit by restricted maximum likelihood when <code>TRUE</code> .
weights	Optional positive numeric case weights, either a vector with one value per row or an expression evaluated in <code>data</code> .
subset	Optional expression selecting rows of <code>data</code> , evaluated in <code>data</code> (as in <code>stats::lm()</code>).
na.action	A function (or function name) controlling missing-value handling, as in <code>lme4::lmer()</code> and <code>stats::lm()</code> . The default is <code>getOption("na.action")</code> (<code>na.omit</code> unless changed). It is applied after <code>subset</code> to the variables the model uses only – the formula variables (fixed and random parts, including evaluated transforms such as <code>log(x)</code>) plus weights and offset – so an NA in an unused column of <code>data</code> never drops a row, exactly like <code>model.frame()</code> . Dropped rows are never silent: a typed <code>mm_rows_dropped</code> message reports how many rows were dropped and which variables had missing values (silence it with <code>mm_control(verbose = -1)</code>). The dropped rows are recorded as <code>na.action(fit)</code> and <code>attr(model.frame(fit), "na.action")</code> , and <code>nobs()</code> counts the rows used. <code>stats::na.omit</code> keeps the

	compact shape; <code>stats::na.exclude</code> pads <code>fitted()</code> , <code>residuals()</code> and in-sample <code>predict()</code> back to the rows of data with NA; <code>stats::na.fail</code> refuses missing values with a typed <code>mm_data_error</code> ; <code>stats::na.pass</code> (or no action configured) is refused with a typed <code>mm_data_error</code> , since a mixed model cannot be fitted to missing values. Factor levels left unused after subset/NA removal are dropped, as <code>model.frame(drop.unused.levels = TRUE)</code> does in <code>lme4</code> .
contrasts	Optional named list of factor contrasts. The engine codes unordered factors with treatment contrasts (<code>contr.treatment</code>) and ordered factors with orthonormal polynomial contrasts (<code>contr.poly</code>), matching <code>lme4/R</code> defaults. A request for any other coding is refused (recode the factor instead).
control	A list from <code>mm_control()</code> .
offset	Optional known component of the linear predictor: a numeric vector with one value per row of data, or an expression evaluated in data (as in <code>lme4::lmer(offset =)</code>). <code>offset()</code> terms in the formula are supported too and are added to it. The engine fits $y - \text{offset}$; <code>fitted()</code> and <code>predict()</code> add the offset back, so they match <code>lmer</code> .

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. The fit checks for a user interrupt (Ctrl-C / Esc) between optimizer evaluations and stops with a typed `mm_interrupted` error (the check happens at evaluation boundaries, so a single very expensive evaluation finishes first). Evaluation budgets are bounded (a bounded budget caps optimizer iterations; it does not prove every native evaluation terminates); 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`.

Formula language

The fixed-effects part accepts R's formula language as `lm()/lmer()` do: `factor()`, `relevel()`, `cut()`, `scale()`, `poly()`, `splines::ns()/bs()`, `as.numeric()`, comparisons such as `I(x > 0)`, `offset()`, and the operators `*`, `:`, `^`, `%in%`, `-` and parenthesised groups such as `(a + b)^2` or `a * (b + c)`. Terms the engine cannot evaluate itself are evaluated in R (with `stats::model.frame()` semantics: on the full data, before `subset/na.action`) and sent to the engine as synthetic columns; coefficient names match `lme4`'s (`poly(x, 2)1`, `factor(cyl)6`), and `predict()` on new data reuses the training basis (`predvars`), as `lm()` does. A logical predictor is coded like a factor with levels `FALSE/TRUE`, as `model.matrix()` codes it. Such transforms are not supported inside random-effect terms, where the engine refuses them with a typed error.

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_allfit

Refit a model with every available optimizer

Description

mm_allfit() is mixeff's counterpart of lme4::allFit(): it refits fit once per optimizer (through [update\(\)](#) with mm_control(optimizer =), keeping the rest of the fit's control) and summarises the fixed effects, covariance parameters, log-likelihood, convergence status and run time side by side, so a fit's sensitivity to the optimizer is visible. An optimizer that is not compiled into this build, or whose refit fails, is reported as a failed row with its error message – never dropped.

Usage

```
mm_allfit(fit, optimizers = mm_allfit_optimizers(), verbose = FALSE)
```

Arguments

fit	A fitted mm_lmm or mm_glm.
optimizers	Character vector of mm_control() optimizer names. The default tries every name mm_control() accepts.
verbose	Print each optimizer as it runs.

Value

An object of class mm_allfit: a named list of fits (or conditions for failed refits). summary() returns a list with which.OK, msgs, fixef, theta, llik, fit_status, and times, mirroring summary(lme4::allFit(...)).

Examples

```
set.seed(1)
d <- data.frame(g = factor(rep(1:8, each = 5)), x = rep(0:4, 8))
d$y <- 1 + 0.5 * d$x + rnorm(8)[d$g] + rnorm(40)
fit <- lmm(y ~ x + (1 | g), d, control = mm_control(verbose = -1))
af <- mm_allfit(fit, optimizers = c("auto", "pattern_search", "cobyla"))
summary(af)$llik
```

mm_bootmer

*Parametric bootstrap of any statistic of a fitted LMM***Description**

mm_bootmer() is mixeff's counterpart of lme4::bootMer(): it simulates nsim responses from the fitted model, refits the model to each, and evaluates FUN on every refit. The result has class c("mm_bootmer", "boot") with the fields boot::boot.ci() reads (t0, t, R, sim, statistic, ...), so boot::boot.ci(b, type = "perc", index = 1) works as it does for a bootMer result.

Usage

```
mm_bootmer(
  fit,
  FUN,
  nsim = 1L,
  seed = NULL,
  type = c("parametric", "semiparametric"),
  use.u = FALSE,
  verbose = FALSE
)
```

Arguments

fit	A fitted mm_lmm. GLMM fits are refused (the GLMM parametric bootstrap is confint(fit, method = "bootstrap")).
FUN	A function of a fitted mm_lmm returning a numeric vector (for example fixef, or function(m) c(fixef(m), sigma = sigma(m))).
nsim	Number of bootstrap replicates.
seed	Optional seed; NULL uses the current R RNG state, so set.seed() governs reproducibility.
type	Only "parametric" is available. lme4's "semiparametric" residual bootstrap is refused.
use.u	FALSE (default) draws new random effects for each replicate (lme4's default, the unconditional bootstrap); TRUE conditions on the fitted random effects and resamples only the residual noise.
verbose	Print a progress dot per replicate.

Details

The function is named mm_bootmer() rather than bootMer() because lme4::bootMer() is not an S3 generic: exporting a bootMer() would mask lme4's for merMod fits.

Replicates use the same model, REML setting, weights, and mm_control() as fit; each refit warm-starts from the fitted covariance parameters (as lme4's refit() does). A replicate whose refit fails, or whose FUN errors, is recorded as an NA row and counted in attr("bootFail") (lme4's convention) – never silently dropped.

Value

An object of class `c("mm_bootmer", "boot")`: a list with `t0` (`FUN(fit)`), `t` (an `nsim` x `length(t0)` matrix), `R`, `data`, `seed`, `statistic`, `sim = "parametric"`, `call`, and `mle`, plus attributes `bootFail` (failed replicates) and `boot.fail.msgs`.

See Also

`confint()` with `method = "bootstrap"` for engine-run fixed-effect bootstrap intervals; `simulate.mm_lmm()`; `refit()`.

Examples

```
set.seed(1)
d <- data.frame(g = factor(rep(1:8, each = 5)), x = rep(0:4, 8))
d$y <- 1 + 0.5 * d$x + rnorm(8)[d$g] + rnorm(40)
fit <- lmm(y ~ x + (1 | g), d, control = mm_control(verbose = -1))
b <- mm_bootmer(fit, fixef, nsim = 20, seed = 2)
apply(b$t, 2, sd)
if (requireNamespace("boot", quietly = TRUE)) {
  boot::boot.ci(b, type = "perc", index = 2)
}
```

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,
  optCtrl = NULL,
  disp_method = NULL,
  disp_dof_correction = NULL,
  max_phi_iter = NULL
)
```

Arguments

verbose	Integer verbosity level. Use -1 to suppress the automatic model explanation and the fit notices (the GLMM estimator notice, the mm_rows_dropped missing-value notice, grouping coercion and scaling advisories).
max_feval	Optional positive integer capping the optimizer's objective evaluations. Most useful for <code>glmm()</code> with <code>method = "joint_laplace"</code> , 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 behavior), "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.
optCtrl	Optional named list in the style of lme4's <code>lmerControl(optCtrl =)</code> , translated to the engine's controls: <code>maxfun/maxeval</code> -> <code>max_feval</code> ; <code>ftol_rel/ftol_abs/xtol_rel</code> -> the arguments of the same name; <code>xtol_abs</code> (scalar or one per theta) -> the absolute parameter tolerance; <code>rhobeg</code> (scalar or one per theta) -> the optimizer's initial step. Unknown names are refused with an <code>mm_arg_error</code> rather than ignored. lme4's <code>check.conv.*</code> options have no counterpart: <code>mixeff</code> does not emit post-hoc convergence warnings; the fit's convergence status is the engine's typed certificate (<code>optimizer_certificate()</code> , <code>verify_convergence()</code>).
disp_method, disp_dof_correction, max_phi_iter	lme4 2.1-0's <code>glmerControl()</code> dispersion controls for <code>glmm()</code> fits with a free dispersion parameter (Gamma, inverse Gaussian, Gaussian with a non-identity link). <code>disp_method = "moment"</code> (the default) profiles the dispersion phi in a damped fixed-point loop around PIRLS, with the working weights divided by phi; "old/buggy" restores lme4 < 2.1 (working weights with phi = 1, theta relative to sigma()). <code>disp_dof_correction</code> (default TRUE) divides the deviance by $n - \text{rank}([X, Z])$ rather than n in the moment estimator, and <code>max_phi_iter</code> (lme4's <code>maxPhiIter</code> , default 100) caps the phi iterations per objective evaluation. Both only matter for "moment". NULL keeps the engine default (the lme4 2.1-0 default). Invalid values raise an <code>mm_arg_error</code> . For families without a free dispersion (binomial, Poisson, negative binomial) and for <code>lmm()</code> the settings have no effect: they are kept on the control, and the fit announces that they were ignored with an <code>mm_control_ignored_notice</code> message (silenced by <code>verbose = -1</code>). lme4 ignores them silently.

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_family

Family objects and prior weights for mixeff fits

Description

`family()` returns the R [family\(\)](#) object of a fit: [gaussian\(\)](#) for [lmm\(\)](#) fits and the GLMM family (with its link) for [glmm\(\)](#) fits. For negative-binomial fits it is `MASS::negative.binomial(theta)` at the fitted (or fixed) `theta`, like `lme4::glmer.nb()`; when MASS is not installed an equivalent family object with the same variance, deviance and link functions is returned.

Usage

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

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

## S3 method for class 'mm_lmm'
weights(object, type = c("prior", "working"), ...)

## S3 method for class 'mm_glmm'
weights(object, type = c("prior", "working"), ...)
```

Arguments

<code>object</code>	A fitted <code>mm_lmm</code> or <code>mm_glmm</code> object.
<code>...</code>	Unused.
<code>type</code>	"prior" or "working" (GLMMs only).

Details

`weights()` mirrors `lme4::weights.merMod()`: `type = "prior"` returns the prior weights, a vector of ones when the model was fitted without weights; `type = "working"` returns the final PIRLS working weights of a GLMM.

Value

A family object, or a numeric vector of weights.

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))
family(fit)
head(weights(fit))
```

mm_formula_manifest	<i>The wrapper's formula manifest</i>
---------------------	---------------------------------------

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?*". Fitted objects do not carry the whole manifest: each one stamps the narrower `fit$schema` record (artifact schema name and version, crate version, package version), which the serialization and reporting paths read back. What lets `audit()` describe a fit after `readRDS()`, with no live Rust handle, is the durable compiled artifact stored beside it.

Usage

```
mm_formula_manifest()
```

Details

Capability flags evolve across releases; 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 `mixeff-rs` 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

<code>fit</code>	A fitted <code>mm_lmm</code> .
<code>specs</code>	Character vector, or a one-sided formula such as <code>~ trt</code> or <code>~ trt group</code> , naming the displayed marginal dimensions.
<code>by</code>	Optional character vector of grouping variables for marginal summaries or pair-wise comparisons.
<code>at</code>	Named list of fixed-predictor values to force in the grid.
<code>cov.reduce</code>	Function used to reduce numeric fixed predictors that are not explicitly gridded.
<code>...</code>	Reserved for future methods.
<code>grid</code>	Optional object returned by <code>mm_grid()</code> .
<code>method</code>	Requested inference method, passed to <code>contrast()</code> .
<code>level</code>	Confidence level for intervals computed from contrast standard errors.
<code>target</code>	Prediction target. Only "population" is implemented.
<code>scale</code>	Prediction scale. Gaussian LMMs have identical "link" and "response" scales.
<code>weights</code>	Averaging weights for <code>mm_means()</code> and <code>mm_comparisons()</code> . "equal" weights reference-grid cells equally; "proportional" weights cells by observed fixed-factor frequencies.
<code>comparison</code>	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.

<code>mm_influence</code>	<i>Leverage, Cook's distance and deletion influence for mixed-effects fits</i>
---------------------------	--

Description

`hatvalues()` returns the diagonal of the hat matrix of the fitted mixed model, `cooks.distance()` the Cook's distances computed from them, and `influence()` refits the model with each case (or each level of a grouping variable) deleted, as `lme4`'s methods of the same names do.

Usage

```
## S3 method for class 'mm_lmm'
hatvalues(model, fullHatMatrix = FALSE, ...)

## S3 method for class 'mm_glmm'
hatvalues(model, fullHatMatrix = FALSE, ...)

## S3 method for class 'mm_lmm'
cooks.distance(model, ...)

## S3 method for class 'mm_glmm'
cooks.distance(model, ...)

## S3 method for class 'mm_lmm'
influence(model, groups = NULL, data = NULL, do.coef = TRUE, ncores = 1L, ...)

## S3 method for class 'mm_glmm'
influence(model, groups = NULL, data = NULL, do.coef = TRUE, ncores = 1L, ...)

## S3 method for class 'mm_influence'
dfbeta(model, which = c("fixed", "var.cov"), ...)

## S3 method for class 'mm_influence'
dfbetas(model, ...)
```

```
## S3 method for class 'mm_influence'
cooks.distance(model, ...)
```

Arguments

model	A fitted mm_lmm or mm_glmm (or, for the dfbeta(), dfbetas() and cooks.distance() methods, an mm_influence result).
fullHatMatrix	If TRUE, return the full $n \times n$ hat matrix instead of its diagonal.
...	Unused.
groups	For influence(): NULL (default) to delete single cases, or the name(s) of grouping column(s) whose levels are deleted one at a time. Columns are looked up in data if given, else in the model frame. Several names delete the levels of their interaction.
data	Optional data frame with one row per observation (in model frame order) used only to look up groups.
do.coef	If FALSE, influence() returns only the hat values without refitting.
ncores	Number of cores for the deletion refits (via parallel::mclapply() on non-Windows platforms); default 1.
which	For dfbeta(): "fixed" (default) for the fixed effects or "var.cov" for the variance-covariance components.

Details

The hat matrix is that of the penalized least-squares problem at the optimum, $H = W^{1/2} C M^{-1} C' W^{1/2}$ with $C = [X, Z \Lambda]$ and $M = C' W C + \text{diag}(\theta_p, I_q)$, where Λ is the relative covariance factor and W the weights. For an LMM, W holds the prior weights and the result equals `lme4::hatvalues()` on the same fit. For a GLMM, W holds the working (IRLS) weights at convergence, as `stats::hatvalues()` does for a glm. For GLMMs with a free dispersion parameter ϕ (Gamma, inverse Gaussian, Gaussian with a non-identity link) the working weights are divided by ϕ , as `lme4 >= 2.1-0`'s `weights(fit, "working")` are, to match the absolute-scale Λ ; `lme4` instead combines the working weights inside the factorization with the prior weights outside it, so the two differ for GLMMs (`lme4` warns that its GLMM hat matrix "may not make sense").

`cooks.distance()` follows `lme4`: $D_i = (r_i / (1 - h_i))^2 h_i / (\phi p)$ with Pearson residuals $r_i ((y - \mu) \sqrt{w} / \sqrt{V(\mu)})$, dispersion ϕ (σ^2 for an LMM, 1 for binomial, Poisson and negative-binomial GLMMs, σ^2 for Gamma, inverse-Gaussian and Gaussian non-identity-link GLMMs) and p the rank of the fixed-effect design.

`influence()` deletes one case (default) or one level of groups at a time and refits with the original settings (REML, family, weights, offset, method, control). Deletion refits that fail are recorded as NA rows with `converged = FALSE` rather than dropped. The result has class `mm_influence` and the component names of `lme4`'s `influence.merMod` result; `stats::dfbeta()`, `stats::dfbetas()` and `stats::cooks.distance()` have methods for it (Cook's distance there is the deletion version based on the change in fixed effects).

Value

`hatvalues()` and `cooks.distance()`: a named numeric vector (or a matrix for `fullHatMatrix = TRUE`). `influence()`: an `mm_influence` list.

Examples

```
set.seed(1)
df <- data.frame(
  y = rnorm(40), x = rnorm(40),
  g = factor(rep(seq_len(8), each = 5))
)
fit <- lmm(y ~ x + (1 | g), df, control = mm_control(verbose = -1))
head(hatvalues(fit))
head(cooks.distance(fit))
infl <- influence(fit, groups = "g")
dfbeta(infl)
cooks.distance(infl)
```

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

Description

Returns the known-schema table that backs [mm_json_negotiate\(\)](#). The wrapper refuses any artifact whose schema name and version are absent from this table, so the table grows as new artifact types are added.

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	<i>Negotiate a JSON schema header against what mixeff supports</i>
-------------------	--

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 <code>schema_name</code> and <code>schema_version</code> as length-1 character strings. Additional fields (e.g., <code>crate_version</code> , <code>package_version</code>) are accepted and ignored at this layer; downstream code records them on the <code>mm_fit</code> 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"))
# Raises mm_schema_error:
try(mm_json_negotiate(list(schema_name = "formula", schema_version = "v99")))
```

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

```
set.seed(1)
d <- data.frame(
  g = factor(rep(1:12, each = 8)),
  trt = factor(rep(c("ctrl", "drug"), 48)),
  x = rnorm(96)
)
d$y <- 1 + 0.5 * (d$trt == "drug") + 0.3 * d$x +
  0.4 * (d$trt == "drug") * d$x + rnorm(12)[d$g] + rnorm(96)
fit <- lmm(y ~ trt * x + (1 | g), d, control = mm_control(verbose = -1))
# Treatment effect at x = 1 (weights use lme4 coefficient names):
mm_lincomb(fit, c(trtdrug = 1, "trtdrug:x" = 1))
```

mm_lmlist

Per-group linear (or generalized linear) model fits

Description

`mm_lmlist()` is `mixeff`'s counterpart of `lme4::lmList()`: it fits `lm()` (or `glm()` when family is given) separately within each level of a grouping factor, using `lme4`'s `response ~ predictors | group` formula. It is named `mm_lmlist()` so attaching `mixeff` does not mask `lme4::lmList()` / `nlme::lmList()`.

Usage

```
mm_lmlist(
  formula,
  data,
  family = NULL,
  subset = NULL,
  weights = NULL,
  na.action = stats::na.omit,
  pool = is.null(family)
)
```

Arguments

formula	A formula $y \sim x \mid g$.
data	A data frame.
family	Optional GLM family; NULL fits <code>lm()</code> .
subset, weights, na.action	Passed to <code>lm()/glm()</code> within each group (subset is applied before splitting). weights may name a column of data.
pool	Pool the residual standard deviation across groups for inference. Defaults to TRUE for <code>lm</code> fits and FALSE for GLMs.

Details

`coef()` returns one row per group (lme4's shape); `confint()` returns a group $\times$ (lower, upper) $\times$ coefficient array. With `pool = TRUE` (the default for `lm` fits, as in lme4) `confint()` uses the residual standard deviation pooled across groups, with the pooled residual degrees of freedom, and `sigma()` returns that pooled value; `confint(x, pool = FALSE)` gives each group's own `lm` t interval (Wald z for GLMs), which is what `confint()` on an `lme4::lmList()` fit returns. A group whose fit fails is kept as NULL and reported, never silently dropped.

Value

An object of class `mm_lmlist`: a named list of `lm/glm` fits (one per group) with attributes `call`, `pool`, `groups`, and `failed`.

Examples

```
set.seed(1)
d <- data.frame(g = factor(rep(1:6, each = 8)), x = rep(0:7, 6))
d$y <- 2 + rnorm(6)[d$g] + (0.5 + rnorm(6, sd = 0.2)[d$g]) * d$x +
  rnorm(48)
ml <- mm_lmlist(y ~ x | g, d)
coef(ml)
confint(ml)[, , "x"]
```

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, ...)

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

## S3 method for class 'mm_lmm'
fixef(object, add.dropped = FALSE, ...)

## S3 method for class 'mm_glmm'
fixef(object, add.dropped = FALSE, ...)

ranef(object, ...)

## Default S3 method:
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, ...)

## Default S3 method:
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, ...)

## 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, fixed.only = FALSE, ...)

## S3 method for class 'mm_glmm'
model.frame(formula, fixed.only = FALSE, ...)

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'
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. For AIC() and BIC() , further fitted models: with several models the result is the data frame stats::AIC() returns (df and AIC/BIC, rows named after the arguments).
add.dropped	For fixef() : if TRUE, coefficients dropped for rank deficiency are included as NA (lme4's add.dropped); by default they are omitted, as in lme4.
condVar	Logical; when TRUE, attach a postVar array of conditional variances to each random-effects table. Gaussian lmm() fits receive the conditional covariances computed by the engine. glmm() fits receive lme4's Laplace-approximation conditional covariances $\sigma^2 * \Lambda (\Lambda' Z' W Z \Lambda + I)^{-1} \Lambda'$ at the fitted modes, with W the final PIRLS working weights. A fit whose conditional-variance computation refuses receives an all-NA postVar plus an mm_unavailable_reason attribute rather than fabricated conditional variances.
REML	Ignored; included for S3 compatibility with likelihood and deviance generics.
k	Penalty per parameter for AIC() .
fixed.only	For model.frame() : return only the fixed-effect variables, as lme4::model.frame.merMod() .
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() .

Details

These methods follow lme4's shapes. [coef\(\)](#) returns, per grouping factor, every fixed-effect column (fixed-only columns repeated) plus the conditional modes, in lme4's column order. [VarCorr\(\)](#) returns lme4's [VarCorr.merMod](#) structure: a named list of covariance matrices (one per random-effect term, so [VarCorr\(fit\)\\$Subject](#) is a matrix) with "stddev" and "correlation" attributes

and list attributes "sc" (sigma()) and "useSc" (TRUE for LMMs; FALSE for every GLMM, as in lme4 >= 2.1-0, so GLMM random-effect SDs are absolute and no Residual row is printed); it prints like lme4 and as.data.frame() gives lme4's long form. mixeff's full-precision long table remains available as VarCorr(fit)\$table (with \$residual_sd). sigma() of a binomial, Poisson or negative-binomial GLMM is 1, as in lme4 (the negative-binomial theta is getME(fit, "glmer.nb.theta")). For GLMMs with a free dispersion parameter (Gamma, inverse Gaussian, and Gaussian with a non-identity link) it is sqrt(phi) with phi = sum(deviance residuals^2) / (n - rank([X, Z])), the dispersion profiled during the fit, as in lme4 >= 2.1-0 (glmerControl() defaults disp_method = "moment", disp_dof_correction = TRUE; lme4 < 2.1-0 used a different, biased estimate). mm_control(disp_dof_correct = FALSE) divides by n instead, and mm_control(max_phi_iter =) caps the phi iterations. With mm_control(disp_method = "old/buggy") the fit reproduces lme4 2.1-0's disp_method = "old/buggy": PIRLS runs with phi = 1, so weights(fit, "working") does not carry 1/phi, and VarCorr() reports theta itself as the random-effect SD, as lme4 2.1-0 does. Under lme4 < 2.1-0 that SD was sigma() * theta. deviance() of a GLMM is the sum of squared deviance residuals, as lme4::deviance.merMod(); -2 * logLik() is the Laplace objective reported in anova(). model.frame() returns lme4's frame (transformed columns such as log(y), a terms attribute, (weights) and (offset) columns); the raw variables the engine used are in fit\$model_frame.

vcov() of a joint-Laplace GLMM is the inverse of the finite-difference Hessian over the fixed effects and covariance parameters, glmer's default (use.hessian = TRUE). When that Hessian is not positive definite (or unavailable) the engine falls back to the fixed-effect block RX of the Laplace penalized least-squares factorization at the optimum, conditional on the covariance parameters – glmer's vcov(fit, use.hessian = FALSE), which glmer also falls back to with a warning. mixeff does the same: the matrix carries attr(, "mm_method") == "laplace_rx_conditional_on_theta", reliability "low" and the engine's notes (attr(, "mm_notes")), and vcov() warns (class mm_vcov_rx_fallback). Such standard errors ignore the uncertainty in the covariance parameters.

Known lme4 2.1-0 discrepancy: for GLMMs with a free dispersion parameter (Gamma, inverse Gaussian, Gaussian with a non-identity link) the penalized least-squares factor already carries the 1 / phi working weights, so mixeff's RX-based vcov() (nAGQ = 0 fits, and the fallback above) is unsc() with no further sigma^2 factor. lme4 >= 2.1-0 still multiplies by sigma()^2 there (and in ranef(condVar = TRUE)), so those values differ from lme4 2.1's by the factor phi = sigma()^2; Hessian-based vcov() values agree.

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 $\theta = \text{NULL}$ (the default) the size parameter is estimated alongside the model, matching `lme4::glmer.nb()`. Supplying a positive θ fits conditional on that value, matching `glmer(family = MASS::negative.binomial(theta))` — which `glmm()` also accepts directly.

Usage

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

Arguments

<code>theta</code>	Optional positive NB2 size (dispersion) parameter. <code>NULL</code> estimates θ 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. 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 <code>format()</code> before parsing.
---------	--

Value

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

Errors

Parse failures are signaled 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))
```

mm_variance_components

Variance decomposition, Nakagawa R2 and ICC for mixed-effects fits

Description

`mm_variance_components()` splits the variance of a fitted mixed model into the components used by Nakagawa and Schielzeth's R^2 and the intraclass correlation: the variance of the fixed-effect linear predictor, the random-effect variance, and the residual (distribution-specific plus observation-level) variance. `mm_r2()` returns the marginal and conditional R^2 and `mm_icc()` the adjusted and unadjusted ICC. The numbers follow `insight::get_variance()`, `performance::r2_nakagawa()` and `performance::icc()` on the equivalent `lme4` fit.

Usage

```
mm_variance_components(fit, tolerance = 1e-05)

mm_r2(fit, tolerance = 1e-05)

mm_icc(fit, tolerance = 1e-05)
```

Arguments

fit	A fitted <code>mm_lmm</code> or <code>mm_glmm</code> .
tolerance	Variances below this value mark the fit as singular.

Details

When the insight and performance packages are installed, mixeff also registers methods so that `performance::icc(fit)`, `performance::r2(fit)` and `insight::get_variance(fit)` work directly on mixeff fits (not by `_group = TRUE`, which needs lme4 internals).

- Fixed variance: `var(X beta)`.
- Random variance: for each random term whose grouping factor has fewer levels than observations, the mean over observations of $z_i' \Sigma z_i$, where z_i holds the term's coefficients' columns of the fixed-effect design (a random slope without a matching fixed column is dropped, as insight does).
- Distribution-specific variance: σ^2 for an LMM; $\pi^2/3$, 1 and $\pi^2/6$ for binomial logit, probit and cloglog links; for Gamma, σ^2 (inverse link) or the log-normal approximation $\log(1 + \sigma^2)$ (log link), with `sigma()` the lme4 $\geq 2.1-0$ dispersion $\sqrt{\phi}$; and the log-normal approximation $\log(1 + V(\mu)/\mu^2)$ for Poisson ($V(\mu) = \mu$) and negative-binomial ($V(\mu) = \mu(1 + \mu/\theta)$) log-link models, with $\mu = \exp(b_0 + v_0/2)$ from the intercept b_0 and the random-effect variance v_0 (observation-level terms excluded) of the null model ($y \sim 1$ with the same random effects), which is refitted. A `cbind(successes, failures)` binomial response divides the binomial value by the mean number of trials. A Gaussian GLMM with a non-identity link uses σ^2 , as insight does for every Gaussian model: note that this is the residual variance on the response scale, while the fixed and random variances are on the link scale, so the R2 and ICC of such a fit depend on the units of the response. Observation-level terms of a Gaussian GLMM do not add to the residual variance (insight treats the model as linear). These follow insight 1.5.
- Inverse-Gaussian GLMMs are refused (`mm_inference_unavailable`, reason code "r2_distribution_variance_unavailable"). insight 1.5 has no inverse-Gaussian branch and falls back to `sigma()` itself (the square root of the dispersion, not a variance); since the inverse-Gaussian dispersion has units of $1/y$, that value, and the R2 and ICC built on it, change when the response is rescaled. `insight::get_variance()` on a mixeff inverse-Gaussian fit therefore returns NA with a warning.
- Observation-level random effects (one level per observation) count towards the residual variance of a GLMM.

If a random-effect variance is below `tolerance` (a singular fit), the random variance is not computed: `mm_r2()` and `mm_icc()` return NA for the quantities that need it and signal a warning, as `performance` does.

Value

`mm_variance_components()`: a named list with `var.fixed`, `var.random`, `var.residual`, `var.distribution`, `var.dispersion`, `var.intercept`, `var.slope` and `cor.slope_intercept` (the layout of `insight::get_variance()`).
`mm_r2()`: a list with `R2_conditional` and `R2_marginal`. `mm_icc()`: a one-row data frame with `ICC_adjusted`, `ICC_conditional` and `ICC_unadjusted`.

Examples

```
set.seed(1)
g <- factor(rep(seq_len(10), each = 6))
df <- data.frame(x = rnorm(60), g = g)
df$y <- 1 + 0.5 * df$x + rnorm(10)[g] + rnorm(60)
```

```
fit <- lmm(y ~ x + (1 | g), df, control = mm_control(verbose = -1))
mm_r2(fit)
mm_icc(fit)
```

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

<code>fit</code>	A fitted <code>mm_fit</code> , usually from <code>lmm()</code> .
<code>sections</code>	Character vector of report sections, or "all".
<code>...</code>	Reserved for future methods.
<code>object</code>	For <code>reporting_table()</code> : a fitted <code>mm_fit</code> , an <code>mm_model_report</code> , or a comparison/test object with a durable ledger.
<code>section</code>	One section name, or "all".
<code>view</code>	"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

<code>object</code>	A compiled <code>mm_spec</code> or fitted <code>mm_fit</code> .
<code>...</code>	Reserved for future methods.

Value

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

parameterization	<i>Inspect covariance parameterization</i>
------------------	--

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	<i>Parametric bootstrap likelihood-ratio comparison</i>
----------------------	---

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,
  threads = 1L,
  ...
)
```

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.
threads	Worker threads for the replicate refits (default 1). The responses are simulated serially and each replicate refits fresh copies of both models, so the result is identical for every value; see bootstrap_control() for the threading contract.
...	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.

plot.mm_fit	<i>Diagnostic plots for mixeff fits</i>
-------------	---

Description

`plot()` on a fitted model draws Pearson residuals against fitted values, the default diagnostic of `lme4`'s `plot.merMod()`, using base graphics. `which = 2` draws a normal quantile-quantile plot of the Pearson residuals instead (also available as `qqnorm(fit)`), and `which = 3` the scale-location plot ($\sqrt{|r|}$ against fitted values).

Usage

```
## S3 method for class 'mm_lmm'
plot(
  x,
  which = 1L,
  smooth = TRUE,
  ask = length(which) > 1L && grDevices::dev.interactive(),
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  ...
)

## S3 method for class 'mm_glmm'
plot(
```

```

    x,
    which = 1L,
    smooth = TRUE,
    ask = length(which) > 1L && grDevices::dev.interactive(),
    main = NULL,
    xlab = NULL,
    ylab = NULL,
    ...
)

## S3 method for class 'mm_lmm'
qqnorm(y, main = NULL, xlab = NULL, ylab = NULL, ...)

## S3 method for class 'mm_glmm'
qqnorm(y, main = NULL, xlab = NULL, ylab = NULL, ...)

```

Arguments

<code>x, y</code>	A fitted <code>mm_lmm</code> or <code>mm_glmm</code> .
<code>which</code>	Integer vector selecting the plots: 1 residuals vs fitted (default), 2 normal Q-Q of the residuals, 3 scale-location.
<code>smooth</code>	If TRUE (default), add a <code>lowess()</code> smooth to plots 1 and 3.
<code>ask</code>	Ask before each new plot when several are drawn on an interactive device.
<code>main, xlab, ylab</code>	Optional titles overriding the defaults.
<code>...</code>	Further graphical parameters passed to <code>graphics::plot()</code> or <code>stats::qqnorm()</code> .

Details

Pearson residuals follow lme4's definition, $(y - \mu) \sqrt{w} / \sqrt{V(\mu)}$ with prior weights w and variance function V ($V = 1$ for an LMM), on the response scale; they are not divided by the residual standard deviation.

Value

`x`, invisibly. `qqnorm()` returns the list from `stats::qqnorm()` invisibly.

See Also

`plot.mm_ranef()` for caterpillar and Q-Q plots of the random effects; `hatvalues.mm_lmm()` for leverage and Cook's distance.

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))

```

```
plot(fit)
plot(fit, which = 2)
qqnorm(fit)
```

plot.mm_ranef

Caterpillar and Q-Q plots of random effects

Description

plot() on the result of ranef() draws a caterpillar plot (dot chart) of the conditional modes for each grouping factor and coefficient, with levels sorted by their value and, when the object carries conditional variances (ranef(fit, condVar = TRUE)), level prediction intervals, mirroring lattice::dotplot() on lme4's ranef() output. qqnorm() draws normal Q-Q plots of the conditional modes, one panel per grouping factor and coefficient.

Usage

```
## S3 method for class 'mm_ranef'
plot(x, level = 0.95, type = c("caterpillar", "qq"), ...)

## S3 method for class 'mm_ranef'
qqnorm(y, ...)
```

Arguments

x, y	An mm_ranef object from ranef().
level	Coverage of the intervals (default 0.95).
type	"caterpillar" (default) or "qq".
...	Further graphical parameters passed to graphics::plot().

Details

When the lattice package is loaded, lattice::dotplot() and lattice::qqmath() also work on ranef() output and return one trellis object per grouping factor, as they do for lme4.

Intervals are omitted for coefficients whose conditional variances are not available (for example GLMM fits where ranef(condVar = TRUE) returns NA variances).

Value

x (or y), invisibly.

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))
re <- ranef(fit, condVar = TRUE)
plot(re)
qqnorm(re)
```

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`, or a subset of the random terms via a one-sided `re.form` formula), 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. Offsets written in the formula (`offset(log(t))`) are evaluated from `newdata`; an `offset =` argument must be supplied again through `predict(offset = )`.

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,
  na.action = stats::na.pass,
  offset = NULL,
  ...
)
```

Arguments

<code>object</code>	A fitted <code>mm_glmm</code> object.
<code>newdata</code>	Optional new data. Must be a <code>data.frame</code> containing every variable the requested prediction needs (the fixed-effect variables, plus the grouping and slope variables of the random terms used); other columns are ignored. Grouping

	columns may be factor, character, or integer (coerced the way they were at fit time); fixed-effect factor and character columns are matched to the training levels, and a level not seen when fitting is refused. Stateful terms (<code>poly()</code> , <code>scale()</code> , <code>ns()</code> , ...) are evaluated with the training basis, as in <code>stats::lm()</code> .
<code>re.form</code>	Random-effects conditioning, following lme4. <code>NULL</code> returns conditional predictions; <code>NA</code> (or <code>~0</code>) returns population-level (fixed-effect) predictions; a one-sided formula naming some of the model's random terms (e.g. <code>~ (1 subject)</code>) conditions on those terms' conditional modes only. Standard errors and intervals are not available for such partial conditioning.
<code>allow.new.levels</code>	When <code>FALSE</code> (default), unseen grouping levels in newdata raise <code>mm_inference_unavailable</code> through the Rust <code>NewReLevels::Error</code> policy. When <code>TRUE</code> , unseen levels are replaced by the population mean (zero random effect), matching <code>lme4::predict(allow.new.levels = TRUE)</code> .
<code>type</code>	Prediction scale. Gaussian LMMs use the same values for "response" and "link". For <code>residuals()</code> , the residual type, as in lme4: LMMs default to "response" ($y - \mu$, also "working"), with "pearson"/"deviance" the weighted residuals $\sqrt{w} * (y - \mu)$; GLMMs default to "deviance" and also offer "pearson" $((y - \mu) * \sqrt{w / V(\mu)})$, "working" and "response".
<code>se.fit</code>	Logical; when <code>TRUE</code> , returns a list with <code>fit</code> and <code>se.fit</code> . For population predictions (<code>re.form = NA</code>) the standard error is the Wald SE of the fixed-effect linear predictor, $\sqrt{\text{diag}(X' V^{-1} X)}$. For conditional predictions (<code>re.form = NULL</code>) 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 <code>allow.new.levels = TRUE</code> — return <code>NA</code> with the engine's reason in the <code>mm_reason</code> attribute. (lme4::predict.merMod offers no conditional SE at all.)
<code>interval</code>	Interval type: "confidence" for the fitted mean or "prediction" for a new observation (adds the residual variance). Population (<code>re.form = NA</code>) intervals are <code>fit +/- z*se</code> computed R-side; conditional (<code>re.form = NULL</code>) bounds come from the engine prediction-variance payload. Returns a matrix with <code>fit/lwr/upr</code> .
<code>level</code>	Confidence level for interval / <code>se.fit</code> intervals.
<code>na.action</code>	Missing-value handling for newdata, as in <code>lme4::predict.merMod()</code> : the default <code>stats::na.pass</code> returns <code>NA</code> for rows with a missing value in a variable the prediction needs; <code>stats::na.omit</code> drops them and <code>stats::na.exclude</code> pads them back as <code>NA</code> . In-sample predictions of a fit made with <code>na.action = na.exclude</code> are padded to the original rows, like <code>fitted()</code> and <code>residuals()</code> .
<code>offset</code>	Offset for newdata rows when the model was fitted with the <code>offset =</code> argument: a numeric vector (one value per newdata row) or an expression evaluated in newdata. Offsets written in the formula (<code>offset(log(t))</code>) are evaluated from newdata automatically. (lme4 silently drops an <code>offset =</code> argument offset for new data; mixeff refuses instead of guessing.)
<code>...</code>	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 `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. For a fit with an offset, conditional SEs/intervals on `newdata` are available on the link scale only.

Known lme4 2.1-0 discrepancy: for GLMMs with a free dispersion parameter (Gamma, inverse Gaussian, Gaussian with a non-identity link) lme4 $\geq$ 2.1-0 builds `predict(se.fit = TRUE)` from `sigma()^2` times its unscaled penalized least-squares covariance, although its PIRLS weights already carry $1 / \phi$. `mixeff`'s SEs use the engine's covariance (`vcov()`), without the extra `sigma^2`, so they differ from lme4 2.1's by roughly a factor of `sigma()`.

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`.

<code>predict.mm_lmm</code>	<i>Predict from a fitted mixeff LMM</i>
-----------------------------	---

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,
  na.action = stats::na.pass,
```

```

    offset = NULL,
    ...
)

## 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_glm'
fitted(object, ...)

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

```

Arguments

<code>object</code>	A fitted <code>mm_lmm</code> object.
<code>newdata</code>	Optional new data. Must be a <code>data.frame</code> containing every variable the requested prediction needs (the fixed-effect variables, plus the grouping and slope variables of the random terms used); other columns are ignored. Grouping columns may be factor, character, or integer (coerced the way they were at fit time); fixed-effect factor and character columns are matched to the training levels, and a level not seen when fitting is refused. Stateful terms (<code>poly()</code> , <code>scale()</code> , <code>ns()</code> , ...) are evaluated with the training basis, as in <code>stats::lm()</code> .
<code>re.form</code>	Random-effects conditioning, following <code>lme4</code> . <code>NULL</code> returns conditional predictions; <code>NA</code> (or <code>~0</code>) returns population-level (fixed-effect) predictions; a one-sided formula naming some of the model's random terms (e.g. <code>~ (1 subject)</code>) conditions on those terms' conditional modes only. Standard errors and intervals are not available for such partial conditioning.
<code>allow.new.levels</code>	When <code>FALSE</code> (default), unseen grouping levels in <code>newdata</code> raise <code>mm_inference_unavailable</code> through the Rust <code>NewReLevels::Error</code> policy. When <code>TRUE</code> , unseen levels are replaced by the population mean (zero random effect), matching <code>lme4::predict(allow.new.levels = TRUE)</code> .
<code>type</code>	Prediction scale. Gaussian LMMs use the same values for "response" and "link". For <code>residuals()</code> , the residual type, as in <code>lme4</code> : LMMs default to

	"response" ($y - \mu$, also "working"), with "pearson"/"deviance" the weighted residuals $\sqrt{w} * (y - \mu)$; GLMMs default to "deviance" and also offer "pearson" ($(y - \mu) * \sqrt{w / V(\mu)}$), "working" and "response".
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, $\sqrt{\text{diag}(X \ V \ X')}$. 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 $\pm z * se$ 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.
na.action	Missing-value handling for newdata, as in lme4::predict.merMod(): the default stats::na.pass returns NA for rows with a missing value in a variable the prediction needs; stats::na.omit drops them and stats::na.exclude pads them back as NA. In-sample predictions of a fit made with na.action = na.exclude are padded to the original rows, like fitted() and residuals().
offset	Offset for newdata rows when the model was fitted with the offset = argument: a numeric vector (one value per newdata row) or an expression evaluated in newdata. Offsets written in the formula (offset(log(t))) are evaluated from newdata automatically. (lme4 silently drops an offset = argument offset for new data; mixeff refuses instead of guessing.)
...	Reserved for generic compatibility.
scaled	Logical; when TRUE, residuals are divided by sigma(object) (once), as in lme4.

Value

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

profile.mm_lmm	<i>Profile a fitted linear mixed model</i>
----------------	--

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, parameter_kind, estimate, lower, upper, regularity, status, reason_code, reason, profiled_criterion). Rows use lme4's names: .sig01, ... (kind "sd" or "cor", numbered in lme4's term order), .sigma, and the fixed effects, all profiled on the ML deviance as lme4 does (a REML fit is refitted by ML); the engine's relative-Cholesky theta1, ... rows follow, profiled on the fit's own criterion. Use confint() with method = "profile" for the matrix form.

Usage

```
## S3 method for class 'mm_lmm'  
profile(fitted, which = NULL, level = 0.95, threads = 1L, ...)
```

Arguments

fitted	A fitted mm_lmm.
which	Optional character vector of parameter names to keep (coefficient names, ".sig01", ".sigma", "theta1", ...).
level	Confidence level for the reported interval endpoints.
threads	Worker threads (default 1); see <code>confint.mm_lmm()</code> .
...	Unused; for generic consistency.

Details

A parameter whose profile is irregular keeps its row: `status` is "non_monotone" (the signed-root deviance is not monotone), "not_bracketing" (the profile does not reach the cutoff on one side), or "failed" (a constrained refit failed); `reason_code` holds the matching regularity code ("non_monotone_profile", "profile_not_bracketing", "profile_failed") and `reason` the engine's explanation. Bounds the profile cannot determine are NA; a bound inside the regular part of the profile is still reported. lme4 instead warns and falls back to linear interpolation (often giving [-1, 1] for a correlation); mixeff reports NA plus the reason rather than an interpolated bound. The other parameters' intervals are unaffected.

Value

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

See Also

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

random_blocks	<i>Inspect random-effect blocks</i>
---------------	-------------------------------------

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	<i>Inspect nearby random-effect spellings for one grouping factor</i>
----------------	---

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 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 model with a new response

Description

refit() fits the same model to a new response, keeping every other setting of the original fit, like lme4::refit(). For an mm_lmm it calls lmm() with the stored model frame, REML setting and prior weights; for an mm_glmm it calls glmm() with the stored family (negative-binomial fits re-estimate theta when the original did), prior weights, offset, method, nAGQ and control settings (verbosity silenced).

Usage

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

## Default S3 method:
refit(object, newresp, ...)

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

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

Arguments

object	A fitted mm_lmm or mm_glmm.
newresp	The new response (see Details).
...	control = to override the stored control settings.

Details

newresp may be a numeric vector, or a one-column data frame such as the output of simulate() with nsim = 1. For binomial GLMMs it may also be a two-column (successes, failures) matrix (the shape simulate() returns for a cbind() response; the trial counts become the new prior weights), a two-level factor, or a logical vector.

As in lme4, newresp is on the scale of the model's response: for a transformed response such as $\log(y) \sim \dots$ it replaces $\log(y)$, and the refit's formula names it .mm_refit_response.

Value

A new fit of the same class as object.

Examples

```

set.seed(1)
df <- data.frame(
  y = rpois(60, 3), x = rnorm(60),
  g = factor(rep(seq_len(10), each = 6))
)
fit <- glmm(y ~ x + (1 | g), df, family = poisson(),
  control = mm_control(verbose = -1))
ystar <- simulate(fit, nsim = 1, seed = 2)
fixef(refit(fit, ystar))

```

REMLcrit

*REML criterion of a linear mixed model***Description**

REMLcrit() mirrors lme4::REMLcrit(): the REML criterion (-2 times the restricted log-likelihood) at the fitted parameters. For a REML fit this is $-2 * \text{logLik}(\text{fit})$. For an ML fit it is the REML criterion evaluated at the ML estimates of the covariance parameters, computed from the penalized least-squares decomposition (`getME(fit, "devcomp")`), exactly as lme4 does. lme4::REMLcrit() is a plain function rather than a generic, so mixeff exports its own; it forwards merMod objects to lme4.

Usage

```
REMLcrit(object)
```

Arguments

object A fitted mm_lmm (or an lme4 merMod).

Value

A single number.

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))
REMLcrit(fit)

```

rePCA

*PCA of the random-effects covariance (lme4's rePCA())***Description**

For each grouping factor, the singular values and left singular vectors of the relative covariance factor (the block-diagonal Cholesky factor of the random-effect covariance divided by the residual variance, lme4's `Lambda`), returned as `prcomp` objects. Near-zero standard deviations flag a random-effect structure that is over-parameterised (singular). `summary()` gives each component's proportion of variance.

Usage

```
rePCA(x)

## Default S3 method:
rePCA(x)

## S3 method for class 'mm_lmm'
rePCA(x)

## S3 method for class 'mm_glmm'
rePCA(x)
```

Arguments

`x` A fitted `mm_lmm` or `mm_glmm` (for GLMM families without a dispersion parameter the covariance itself is used, as in lme4).

Details

Singular values match `lme4::rePCA()`; the rotation columns are eigenvectors of the same matrix and may differ from lme4's in sign.

Value

An object of class `c("mm_prcomplist", "prcomplist")`: a named list (one entry per grouping factor) of `prcomp` objects.

Examples

```
set.seed(1)
d <- data.frame(g = factor(rep(1:10, each = 6)), x = rep(0:5, 10))
d$y <- 1 + 0.5 * d$x + rnorm(10)[d$g] + rnorm(10, sd = 0.3)[d$g] * d$x +
  rnorm(60)
fit <- lmm(y ~ x + (x | g), d, control = mm_control(verbose = -1))
summary(rePCA(fit))
```

reproducibility	<i>Inspect reproducibility metadata</i>
-----------------	---

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	<i>Revive a serialized mixeff object</i>
--------	--

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. Revival recreates the lazy R-side cache, keeps a live native handle, and clears a dead one (an external pointer restored by readRDS()); it does not refit the model. Without a live handle, computations that need the engine model (contrasts, summary() tests, predictions with intervals, profiles, bootstraps) refit it from the stored model frame, with identical results.

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.

<code>roles</code>	<i>Declare or inspect design roles</i>
--------------------	--

Description

`roles()` has two 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.

<code>simulate.mm_lmm</code>	<i>Simulate responses from a mixeff fit</i>
------------------------------	---

Description

Draws responses from a fitted `mm_lmm` or `mm_glmm` following the semantics of `lme4::simulate.merMod()`:

Usage

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

## S3 method for class 'mm_glmm'
simulate(object, nsim = 1, seed = NULL, use.u = FALSE, re.form = NA, ...)
```

Arguments

object	A fitted mm_lmm or mm_glmm.
nsim	Number of simulated responses.
seed	Optional random seed.
use.u	Logical; TRUE is the same as re.form = NULL and FALSE the same as re.form = NA. Specify at most one of use.u and re.form.
re.form	NA or ~ 0 (default NA) to simulate new random effects, NULL to condition on the fitted random effects, or a formula such as $\sim (1 g)$ to condition on those terms' fitted random effects and draw new ones for the others (as lme4).
...	Reserved; lme4 arguments that cannot be honoured (newdata, newparams, ...) are refused with a typed error.

Details

- re.form = NA (the default) or ~ 0 , equivalently use.u = FALSE, draws **new** random effects from the fitted random-effect covariance for every simulation (unconditional, parametric-bootstrap simulation);
- re.form = NULL, equivalently use.u = TRUE, conditions on the fitted conditional modes (BLUPs) and simulates only the response noise around fitted(object).

Partial random-effect formulas are refused with a typed mm_inference_unavailable error.

LMM responses are Gaussian with standard deviation $\sigma / \sqrt{w}$ for prior weights w (lme4 ignores the prior weights here). GLMM responses are family-aware: binomial draws use the prior weights as trial counts and return proportions (y / w , 0/1 values for Bernoulli fits), or a two-column (successes, failures) matrix per simulation when the model was fitted with a cbind() response, as lme4 does; Poisson and negative-binomial (fitted theta) draws return counts. For the families with a free dispersion $\phi = \sigma(\text{object})^2$ the draws follow the fitted variance function: Gamma draws use shape w / ϕ ($\text{Var}(y) = \phi \mu^2 / w$), inverse-Gaussian draws use `statmod::rinvgauss()` with shape w / ϕ ($\text{Var}(y) = \phi \mu^3 / w$; needs the statmod package), and Gaussian non-identity-link draws have standard deviation $\sigma / \sqrt{w}$. New random effects are drawn on the absolute scale of `VarCorr()`, as lme4 $\geq 2.1-0$ does (its theta is the absolute random-effect SD for every GLMM).

Compared with lme4 2.1-0's `simulate.merMod()`: Gamma draws and Gaussian draws without prior weights agree draw for draw (same RNG use) when the estimates agree. lme4 ignores prior weights for Gaussian responses (with a warning); mixeff uses them, as for LMMs. For the inverse Gaussian lme4 passes shape $w / \sigma(\text{object}) - \sqrt{\phi}$, not ϕ – so its draws have variance $\sigma \mu^3 / w$ rather than the fitted $\phi \mu^3 / w$; mixeff uses the fitted variance, as `stats::inverse.gaussian()`\$simulate does. A binomial fit whose response was a factor is simulated as 0/1 numeric (lme4 returns a factor).

The random-number stream is consumed in lme4's order (all random-effect draws first, terms in lme4's internal order, then the response draws), so when the estimates agree a seeded `simulate()` reproduces lme4's draws. Unlike lme4, a supplied seed does not leave the global RNG state changed.

Value

A data frame with one column per simulation (sim_1, ...), rows named like the model frame. For a binomial `cbind()` fit each column is a two-column matrix. Attributes `seed`, `mm_method` and `mm_re_form` record the seed, the simulation route, and the resolved conditioning ("unconditional", "conditional" or "partial"). Following `stats::simulate`, attribute "seed" is the `.Random.seed` in force before simulation when `seed = NULL`, otherwise seed with the RNG kind as attribute "kind".

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))
head(simulate(fit, nsim = 2, seed = 1))
# condition on the fitted random effects
head(simulate(fit, nsim = 1, seed = 1, re.form = NULL))

df$count <- rpois(60, 2)
gfit <- glmm(count ~ x + (1 | g), df, family = poisson(),
  control = mm_control(verbose = -1))
head(simulate(gfit, nsim = 2, seed = 1))
```

step

Stepwise model selection

Description

`stats::step()` is not a generic, so `mixeff` (like `lmerTest`) exports a `step()` generic whose default method is `stats::step()`; the `mm_lmm` method is the `lmerTest`-style backward elimination documented in [step.mm_lmm\(\)](#).

Usage

```
step(object, ...)

## Default S3 method:
step(object, ...)
```

Arguments

```
object      A fitted model.
...         Passed to the method (for the default, to stats::step()).
```

Value

See the method.

step.mm_lmm	<i>Backward elimination of random and fixed effects (lmerTest's step())</i>
-------------	---

Description

step() for mm_lmm fits follows lmerTest::step(): it first removes random-effect terms one at a time while the least significant has $p > \alpha_{\text{random}}$, then removes fixed-effect terms (respecting marginality: a term is only dropped when no higher-order term contains it) while the least significant has $p > \alpha_{\text{fixed}}$, refitting after every removal.

Usage

```
## S3 method for class 'mm_lmm'
step(
  object,
  ddf = c("Satterthwaite", "Kenward-Roger"),
  alpha.random = 0.1,
  alpha.fixed = 0.05,
  reduce.fixed = TRUE,
  reduce.random = TRUE,
  keep,
  ...
)
```

Arguments

object	A fitted mm_lmm.
ddf	Denominator degrees of freedom for the fixed-effect F tests.
alpha.random, alpha.fixed	Elimination thresholds.
reduce.fixed, reduce.random	Whether to eliminate fixed / random terms.
keep	Character vector of fixed-effect terms never to remove.
...	Unused.

Details

Differences from lmerTest, by design: random terms are tested with `test_random_effect()`'s boundary-corrected likelihood-ratio test (a chi-bar-square mixture, so p-values are about half of lmerTest's `ranova()` naive chi-square); only whole random terms are removed (lmerTest's `ranova()` also tries reducing $(x | g)$ to $(1 | g)$ – write the reduced term as its own term, e.g. $(1 | g) + (0 + x | g)$, to make that reduction available); and the last random term is never removed (a model without random effects is not an LMM). Fixed terms are tested with `test_effect()` (ddf = "Satterthwaite" or "Kenward-Roger").

Value

An object of class `mm_step` with the elimination tables `random` and `fixed` and the final model in `$model` (also returned by `lmerTest::get_model()` when `lmerTest` is loaded).

Examples

```
set.seed(1)
d <- data.frame(g = factor(rep(1:10, each = 6)), x = rep(0:5, 10),
                z = rnorm(60))
d$y <- 1 + 0.5 * d$x + rnorm(10)[d$g] + rnorm(60)
fit <- lmm(y ~ x + z + (1 | g) + (0 + x | g), d,
          control = mm_control(verbose = -1))
s <- step(fit)
s
s$model
```

supported_models	<i>The supported model envelope</i>
------------------	-------------------------------------

Description

`supported_models()` returns the package's machine-readable support registry: which GLMM family/link/estimator combinations are supported, which features are supported, refused, or carry mixeff-specific semantics, and the package's explicit non-goals. The registry ships as `inst/support/model-support.json` and is the single source of truth for the migration vignette's support table; the test suite asserts that the fitting code enforces exactly this envelope (unsupported combinations refuse with the declared condition classes and reason codes).

Usage

```
supported_models()
```

Details

`mixeff` aims for tested statistical agreement with `lme4` on this envelope — a documented, closed surface — not blanket equivalence with everything `lmer()`/`glmer()` accept.

Value

A list of class `mm_model_support` with elements `glmm_families` (data frame: family, links, estimators, notes), `features` (data frame: feature, lmm, glmm, notes), `non_goals` (character), and `registry` (the parsed JSON, verbatim).

Examples

```
supported_models()
```

test_effect	<i>Test a fixed-effect term</i>
-------------	---------------------------------

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	<i>Test a random-effect variance component</i>
--------------------	--

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 parameter and reports the Self-Liang 50:50 mixture reference distribution; a comparison that adds only correlation parameters (e.g. `| |` versus `|`) is referred to the ordinary chi-square, since a zero correlation is interior to the parameter space (`method = "boundary_lrt_ordinary_chisq"`). 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

<code>fit</code>	A fitted <code>mm_lmm</code> .
<code>term</code>	Random-effect term to test. This can be the term id (<code>"r0"</code>), the original random-effect fragment such as <code>"(1 subject)"</code> , or a unique grouping factor name such as <code>"subject"</code> .
<code>method</code>	Currently <code>"boundary_lrt"</code> .
<code>refit_for_comparison</code>	How to handle REML fits. <code>"auto"</code> and <code>"ml"</code> refit to ML; <code>"error"</code> refuses.
<code>...</code>	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::update()` for the common cases: changing the formula ($. \sim . - x$), 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 <code>mm_lmm</code> or <code>mm_glmm</code> .
formula.	A formula-change applied with <code>stats::update.formula()</code> ; omit to keep the current formula. Random-effect terms ($(x g)$, $(x g)$) are preserved across $. \sim .$ edits.
...	Arguments to override on the re-fit. For <code>mm_lmm</code> : <code>data</code> , <code>REML</code> , <code>weights</code> , <code>offset</code> , <code>na.action</code> , <code>control</code> . For <code>mm_glmm</code> : additionally <code>family</code> , <code>method</code> , <code>nAGQ</code> , <code>inference</code> . The fit's missing-value handling carries over: a refit of the stored model frame keeps the original <code>na.action</code> record (so <code>na.exclude</code> padding persists), and new data is filtered with the original <code>na.action</code> (<code>na.omit</code> or <code>na.exclude</code>) unless <code>na.action</code> is supplied.
evaluate	If <code>TRUE</code> (default) re-fit and return the new model; if <code>FALSE</code> 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	<i>Verify convergence of a fitted mixed model</i>
--------------------	---

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 (LMM-only, 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
)

## S3 method for class 'mm_glmm'
```

```

verify_convergence(
  fit,
  ...,
  restart = TRUE,
  jitter_starts = 1L,
  jitter_scale = 1e-04,
  consensus = FALSE,
  max_feval = 1000L,
  objective_tolerance = 1e-04,
  theta_tolerance = 0.001,
  beta_tolerance = 0.001
)

```

Arguments

<code>fit</code>	A fitted <code>mm_lmm</code> from <code>lmm()</code> or <code>mm_glmm</code> from <code>glmm()</code> .
<code>...</code>	Reserved for future methods.
<code>restart</code>	Logical; re-optimize starting from the fitted optimum and compare against it.
<code>jitter_starts</code>	Number of restarts from jittered copies of the fitted covariance parameters.
<code>jitter_scale</code>	Relative scale of the jitter applied to theta.
<code>consensus</code>	Logical; also refit with an engine-chosen alternate optimizer and compare. Default FALSE: this vendored build compiles without the optional <code>nlopt</code> backend, and for some models the engine's alternate choice is an <code>nlopt</code> optimizer — its absence would then be reported as a non-agreeing run (status <code>fragile</code>) that reflects the build, not the fit. Enable it when you want the consensus pass and will read the per-run diagnostics.
<code>max_feval</code>	Positive integer cap on objective evaluations per verification run.
<code>objective_tolerance</code> , <code>theta_tolerance</code> , <code>beta_tolerance</code>	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 $2 + \text{jitter_starts}$ fits (plus consensus runs when enabled). GLMM verification uses wider default tolerances than the LMM route (objective $1e-4$, beta $1e-3$): each objective evaluation carries inner-PIRLS noise, and the joint path's derivative-free beta search bounds beta reproducibility at its `ftol` stop. A GLMM whose requested `joint_laplace` estimator was substituted by the labelled fast-PIRLS fallback (see `optimizer_certificate()`) re-requests the joint route; when the verification refit falls back the same way — the expected case — every run verifies the profiled objective the fitted numbers came from, with ordinary objective deltas. Runs are only reported as substitution runs (no objective delta) in the asymmetric case where the reference refit certifies the joint route but an individual verification run falls back.

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

```
set.seed(1)
df <- data.frame(t = rep(0:4, 8), s = factor(rep(1:8, each = 5)))
df$y <- 1 + 0.5 * df$t + rnorm(8)[df$s] + rnorm(40)
fit <- lmm(y ~ t + (1 | s), df, control = mm_control(verbose = -1))
verify_convergence(fit)
```

Index

AIC(), 50
AIC.mm_glmm (mm_lmm-methods), 47
AIC.mm_lmm (mm_lmm-methods), 47
anova.mm_glmm, 4
anova.mm_lmm, 4
as.data.frame.mm_ranef
(mm_lmm-methods), 47
as.data.frame.mm_varcorr
(mm_lmm-methods), 47
as_json, 6
audit, 6
audit(), 8, 13, 21, 36
audit_design, 8

BIC(), 50
BIC.mm_glmm (mm_lmm-methods), 47
BIC.mm_lmm (mm_lmm-methods), 47
binomial(), 24
bootstrap_control, 8
bootstrap_control(), 11, 14, 15, 17, 58, 77

changes, 9
coef.mm_glmm (mm_lmm-methods), 47
coef.mm_lmm (mm_lmm-methods), 47
compare, 10
compare(), 4, 5
compare_covariance, 11
compile_model, 12
compile_model(), 7, 11, 21, 67
confint(), 33, 65, 66
confint.mm_glmm, 13
confint.mm_lmm, 14
confint.mm_lmm(), 14, 66
contrast (contrast.mm_glmm), 16
contrast(), 4, 39, 40, 45
contrast.mm_glmm, 16
cooks.distance.mm_glmm (mm_influence),
40
cooks.distance.mm_influence
(mm_influence), 40

cooks.distance.mm_lmm (mm_influence), 40

deviance.mm_glmm (mm_lmm-methods), 47
deviance.mm_lmm (mm_lmm-methods), 47
df.residual.mm_glmm (mm_lmm-methods), 47
df.residual.mm_lmm (mm_lmm-methods), 47
df_for_contrast, 17
df_for_contrast(), 44, 45
dfbeta.mm_influence (mm_influence), 40
dfbetas.mm_influence (mm_influence), 40
diagnostics, 18
drop1(), 4
drop1.mm_glmm, 19
drop1.mm_lmm, 19

estimability, 20
explain_model, 21
explain_model(), 11, 29, 33
extractAIC(), 50
extractAIC.mm_glmm (mm_lmm-methods), 47
extractAIC.mm_lmm (mm_lmm-methods), 47

family(), 35
family.mm_glmm (mm_family), 35
family.mm_lmm (mm_family), 35
fit_handle_alive, 22
fit_status (diagnostics), 18
fitted(), 30, 62, 65
fitted.mm_glmm (predict.mm_lmm), 63
fitted.mm_lmm (predict.mm_lmm), 63
fixef (mm_lmm-methods), 47
formula.mm_glmm (mm_lmm-methods), 47
formula.mm_lmm (mm_lmm-methods), 47

Gamma(), 24
gaussian(), 24, 35
getME, 22
glmm, 24
glmm(), 7, 12, 22, 28, 33–35, 50, 52, 68, 81
graphics::plot(), 59, 60

- hatvalues.mm_glmm(mm_influence), 40
- hatvalues.mm_lmm(mm_influence), 40
- hatvalues.mm_lmm(), 59
- inference_options, 26
- inference_table, 27
- influence.mm_glmm(mm_influence), 40
- influence.mm_lmm(mm_influence), 40
- inverse.gaussian(), 24
- is_singular, 28
- lme4::lmer(), 29
- lmm, 29
- lmm(), 7, 12, 22, 24, 25, 28, 33–35, 47, 50, 56, 68, 81
- logLik.mm_glmm(mm_lmm-methods), 47
- logLik.mm_lmm(mm_lmm-methods), 47
- mm_allfit, 31
- mm_bootmer, 32
- mm_comparisons(mm_grid), 37
- mm_control, 33
- mm_control(), 25, 30–32, 82
- mm_family, 35
- mm_formula_manifest, 36
- mm_formula_manifest(), 43
- mm_glmm, 79
- mm_grid, 37
- mm_icc(mm_variance_components), 53
- mm_influence, 40
- mm_json_known_schemas, 42
- mm_json_known_schemas(), 43
- mm_json_negotiate, 43
- mm_json_negotiate(), 42
- mm_lincomb, 44
- mm_lmllist, 45
- mm_lmm, 79
- mm_lmm-methods, 47
- mm_means(mm_grid), 37
- mm_negative_binomial, 52
- mm_negative_binomial(), 24
- mm_parse_formula, 52
- mm_predictions(mm_grid), 37
- mm_r2(mm_variance_components), 53
- mm_variance_components, 53
- model.frame(), 79
- model.frame.mm_glmm(mm_lmm-methods), 47
- model.frame.mm_lmm(mm_lmm-methods), 47
- model.matrix.mm_glmm(mm_lmm-methods), 47
- model.matrix.mm_lmm(mm_lmm-methods), 47
- model_report, 55
- ngrps(mm_lmm-methods), 47
- nobs(), 29
- nobs.mm_glmm(mm_lmm-methods), 47
- nobs.mm_lmm(mm_lmm-methods), 47
- optimizer_certificate, 56
- optimizer_certificate(), 34, 35, 82
- parameterization, 57
- parametric_bootstrap, 57
- plot.mm_fit, 58
- plot.mm_glmm(plot.mm_fit), 58
- plot.mm_lmm(plot.mm_fit), 58
- plot.mm_ranef, 60
- plot.mm_ranef(), 59
- poisson(), 24
- predict(), 30
- predict.mm_glmm, 61
- predict.mm_lmm, 63
- profile.mm_lmm, 65
- profile.mm_lmm(), 15
- qqnorm.mm_glmm(plot.mm_fit), 58
- qqnorm.mm_lmm(plot.mm_fit), 58
- qqnorm.mm_ranef(plot.mm_ranef), 60
- random_blocks, 66
- random_options, 67
- random_options(), 11, 27
- ranef(mm_lmm-methods), 47
- ranef(), 60
- refit, 68
- refit(), 33
- REMLcrit, 69
- rePCA, 70
- reporting_table(model_report), 55
- reproducibility, 71
- residuals(), 30, 62, 65
- residuals.mm_glmm(predict.mm_lmm), 63
- residuals.mm_lmm(predict.mm_lmm), 63
- revive, 71
- revive(), 6, 22
- roles, 72
- sigma.mm_glmm(mm_lmm-methods), 47

`sigma.mm_lmm` (`mm_lmm-methods`), 47
`simulate()`, 68
`simulate.mm_glm` (`simulate.mm_lmm`), 72
`simulate.mm_lmm`, 72
`simulate.mm_lmm()`, 33
`stats::cooks.distance()`, 41
`stats::dfbeta()`, 41
`stats::dfbetas()`, 41
`stats::glm()`, 25
`stats::hatvalues()`, 41
`stats::lm()`, 29, 62, 64
`stats::model.frame()`, 30
`stats::na.exclude`, 30, 62, 65
`stats::na.fail`, 30
`stats::na.omit`, 29, 62, 65
`stats::na.pass`, 30, 62, 65
`stats::qqnorm()`, 59
`stats::update()`, 79
`stats::update.formula()`, 79
`step`, 74
`step.mm_lmm`, 75
`step.mm_lmm()`, 74
`summary()`, 4
`supported_models`, 76

`terms.mm_glm` (`mm_lmm-methods`), 47
`terms.mm_lmm` (`mm_lmm-methods`), 47
`test_effect`, 77
`test_effect()`, 75, 78
`test_random_effect`, 78
`test_random_effect()`, 75

`update()`, 31
`update.mm`, 79
`update.mm_glm` (`update.mm`), 79
`update.mm_lmm` (`update.mm`), 79

`VarCorr` (`mm_lmm-methods`), 47
`vcov()`, 50
`vcov.mm_glm` (`mm_lmm-methods`), 47
`vcov.mm_lmm` (`mm_lmm-methods`), 47
`verify_convergence`, 80
`verify_convergence()`, 22, 34

`weights.mm_glm` (`mm_family`), 35
`weights.mm_lmm` (`mm_family`), 35