Package: mixpower (via r-universe)

November 6, 2026

Title Simulation-Based Power Analysis for Mixed-Effects Models

Version 1.2.0

Description A comprehensive, simulation-based toolkit for power and sample-size analysis for linear and generalized linear mixed-effects models (LMMs and GLMMs). Supports Gaussian, binomial, Poisson, and negative binomial families via 'lme4'; Wald and likelihood-ratio tests; multi-parameter sensitivity grids; power curves and minimum sample-size solvers; parallel evaluation with deterministic seeds; and full reproducibility (manifests, result bundling, and export to CSV/JSON). Delivers thorough diagnostics per run (failure rate, singular-fit rate, effective N) and publication-ready summary tables. References: Bates et al. (2015) ``Fitting Linear Mixed-Effects Models Using lme4'' <doi:10.18637/jss.v067.i01>; Green and MacLeod (2016) ``SIMR: an R package for power analysis of generalized linear mixed models by simulation'' <doi:10.1111/2041-210X.12504>.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports stats, lme4

Suggests testthat (>= 3.0.0), knitr, rmarkdown, digest, jsonlite, glmmTMB, lmerTest, pbkrtest, dplyr, tidyr, ggplot2, tibble

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/alitovchenko/mixpower, https://CRAN.R-project.org/package=mixpower

BugReports https://github.com/alitovchenko/mixpower/issues

Config/pak/sysreqs cmake make

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

Date/Publication 2026-06-27 18:17:19 UTC

RemoteUrl https://github.com/alitovchenko/mixpower

RemoteRef HEAD

RemoteSha afdfbd3feea58d94665a84ecd9c9238da1736767

Contents

mixpower-package . . . 3
as_tibble.mp_power . . . 5
autoplot.mp_sensitivity . . . 6
effect_size . . . 6
fit_model . . . 8
mp_assumptions . . . 8
mp_attach_calibration . . . 10
mp_backend . . . 10
mp_backend_glmmtmb . . . 11
mp_backend_lme4 . . . 12
mp_backend_lme4_binomial . . . 13
mp_backend_lme4_nb . . . 14
mp_backend_lme4_poisson . . . 15
mp_bundle_results . . . 15
mp_calibrate . . . 16
mp_compare_models . . . 18
mp_custom_dgp . . . 19
mp_design . . . 20
mp_elicit . . . 22
mp_extend . . . 23
mp_from_fit . . . 24
mp_grid_sample_size . . . 25
mp_manifest . . . 26
mp_methods_text . . . 27
mp_missing . . . 27
mp_plan . . . 29
mp_power . . . 30
mp_power_adaptive . . . 32
mp_power_checkpoint . . . 33
mp_power_curve . . . 34
mp_power_curve_parallel . . . 35
mp_quick_power . . . 36
mp_recommend_method . . . 38
mp_report . . . 39
mp_report_table . . . 40
mp_safeguard_effect . . . 41
mp_scenario . . . 42

mp_scenario_glmmtmb_lmm . . . 43
mp_scenario_lme4 . . . 44
mp_scenario_lme4_binomial . . . 45
mp_scenario_lme4_nb . . . 46
mp_scenario_lme4_poisson . . . 47
mp_sensitivity . . . 48
mp_sensitivity_parallel . . . 48
mp_sesoi . . . 50
mp_solve_sample_size . . . 51
mp_stop . . . 52
mp_write_report . . . 53
mp_write_results . . . 53
plot.mp_power . . . 54
plot.mp_power_curve . . . 54
plot.mp_sensitivity . . . 55
plot_power . . . 55
run_parallel . . . 56
simulate_glmm_binomial_data . . . 56
simulate_glmm_nb_data . . . 57
simulate_glmm_poisson_data . . . 58
simulate_power . . . 58
summarize_simulations . . . 59
test_effect . . . 59
validate_mp_backend . . . 60

Index **61**

`mixpower-package` *Simulation-Based Power Analysis for Mixed-Effects Models*

Description

mixpower is a simulation-based toolkit for power and sample-size analysis for linear and generalized linear mixed-effects models (LMMs and GLMMs). It is design-first (no pilot data required), supports Gaussian, binomial, Poisson, and negative binomial families via **lme4**; Wald and likelihood-ratio tests; multi-parameter sensitivity grids; power curves and minimum sample-size solvers; parallel evaluation with deterministic seeds; and full reproducibility (manifests, result bundling, export to CSV/JSON). Every run reports diagnostics (failure rate, singular-fit rate, effective N).

Details

Typical workflow: (1) define a design with `mp_design()` (cluster sizes, trials per cell) and effect-size assumptions with `mp_assumptions()`; (2) build a scenario with a backend constructor (e.g. `mp_scenario_lme4()` for Gaussian LMM); (3) run `mp_power()` for a single power estimate, `mp_sensitivity()` to vary parameters, or `mp_power_curve()` / `mp_solve_sample_size()` for curves and sample-size. Use a fixed `seed` for reproducibility. Failure and singular-fit rates are always reported and never suppressed.

Getting started

After loading the package, define a design and assumptions, build an lme4 scenario, then call `mp_power()`. See Examples below for a minimal runnable workflow.

Function overview

Design and assumptions: `mp_design()`, `mp_assumptions()`.

Scenarios (LMM/GLMM): `mp_scenario()`, `mp_scenario_lme4()`, `mp_scenario_lme4_binomial()`, `mp_scenario_lme4_poisson()`, `mp_scenario_lme4_nb()`.

Power and sensitivity: `mp_power()` (optional `aggregate = "streaming"`), `mp_sensitivity()`, `mp_sensitivity_parallel()`, `mp_power_curve()`, `mp_power_curve_parallel()`, `mp_solve_sample_size()`, `mp_grid_sample_size()`, `mp_quick_power()`.

Backends and simulators: `mp_backend()`, `validate_mp_backend()`, `mp_backend_lme4()`, `mp_backend_lme4_binomial` `mp_backend_lme4_poisson()`, `mp_backend_lme4_nb()`, `mp_backend_glmmtmb()`, `mp_scenario_glmmtmb_lmm()`, `simulate_glmm_binomial_data()`, `simulate_glmm_poisson_data()`, `simulate_glmm_nb_data()`.

Optional tidyverse (Suggests): `tibble::as_tibble()` and `ggplot2::autoplot()` methods.

Reproducibility and reporting: `mp_manifest()`, `mp_bundle_results()`, `mp_report_table()`, `mp_write_results()`.

Plotting: `plot()` for `mp_power_curve` and `mp_sensitivity` objects.

Getting the most out of mixpower

- Use a fixed `seed` (e.g. `seed = 123`) so runs are reproducible.
- Check `failure_rate` and `singular_rate` in results; investigate if high.
- For nested model comparison use LRT with an explicit `null_formula`.
- Use `mp_power_curve()` or `mp_solve_sample_size()` to choose sample size.
- Vignettes give step-by-step guides: `vignette("mixpower-intro", package = "mixpower")`, `vignette("mixpower-design", package = "mixpower")`, `vignette("mixpower-simulations", package = "mixpower")`, `vignette("mixpower-diagnostics", package = "mixpower")`, `vignette("mixpower-re` `package = "mixpower")`, `vignette("mixpower-extending", package = "mixpower")`.

Author(s)

Maintainer: Alex Litovchenko <al4877@columbia.edu>

References

Bates D, Maechler M, Bolker B, Walker S (2015). "Fitting Linear Mixed-Effects Models Using lme4." *Journal of Statistical Software*, 67(1), 1–48. doi:10.18637/jss.v067.i01.

Green P, MacLeod CJ (2016). "SIMR: an R package for power analysis of generalized linear mixed models by simulation." *Methods in Ecology and Evolution*, 7(4), 493–498. doi:10.1111/2041210X.12504.

See Also

Entry points: `mp_design()`, `mp_power()`, `mp_quick_power()`. Vignettes: `vignette(package = "mixpower")`.

Examples

```
# Minimal workflow: design -> assumptions -> scenario -> power
d <- mp_design(clusters = list(subject = 30), trials_per_cell = 4)
a <- mp_assumptions(
  fixed_effects = list(`(Intercept)` = 0, condition = 0.3),
  residual_sd = 1
)
scn <- mp_scenario_lme4(y ~ condition + (1 | subject), design = d, assumptions = a)
res <- mp_power(scn, nsim = 50, seed = 123)
summary(res)
```

---

as_tibble.mp_power *Coerce mixpower results to a tibble*

Description

Requires the **tibble** package (installed with the tidyverse).

Usage

```
## S3 method for class 'mp_power'
as_tibble(x, ...)
```

Arguments

`x`	An `mp_power`, `mp_sensitivity`, or `mp_power_curve` object.
`...`	Passed to `tibble::as_tibble()` on the underlying data frame.

Value

A `tibble::tibble()`.

autoplot.mp_sensitivity *ggplot2 diagnostic plot for sensitivity or power curve*

Description

Requires **ggplot2**. Intended as an optional alternative to base `plot.mp_sensitivity()` / `plot.mp_power_curve()`.

Usage

```
## S3 method for class 'mp_sensitivity'
autoplot(
  object,
  ...,
  y = c("estimate", "failure_rate", "singular_rate", "n_effective")
)
```

Arguments

`object`	An `mp_sensitivity` or `mp_power_curve` object.
`...`	Unused; reserved for consistency with ggplot2 generics.
`y`	For sensitivity/curve objects: same as `plot()` — `"estimate"`, `"failure_rate"`, `"singular_rate"`, or `"n_effective"`.

Value

A **ggplot2** object.

effect_size *Effect-size converters for eliciting assumptions*

Description

Helpers that translate standardized or published effect sizes into the raw coefficients and variance components that `mp_assumptions()` expects, so you do not have to hand-pick regression coefficients. They compose directly:

Usage

```
mp_d_to_beta(d, sd = 1)

mp_beta_to_d(beta, sd = 1)

mp_r2_to_beta(r2, sd_resid = 1, predictor_sd = 1)
```

```
mp_beta_to_r2(beta, sd_resid = 1, predictor_sd = 1)

mp_icc_to_sd(icc, sd_resid = 1)

mp_sd_to_icc(sd, sd_resid = 1)

mp_or_to_logodds(or)

mp_logodds_to_or(beta)

mp_t_to_beta(t, se)

mp_f_to_beta(f, se)
```

Arguments

d	Cohen's d (standardized mean difference).
sd	Standard deviation defining the standardization (e.g. the total outcome SD `sqrt(tau^2 + sigma^2)`).
beta	A raw coefficient.
r2	Target (partial) proportion of variance explained, in `[0, 1)`.
sd_resid	Residual standard deviation.
predictor_sd	Standard deviation of the predictor (default 1).
icc	Target intraclass correlation, in `[0, 1)`.
or	Odds ratio (> 0).
t	A t statistic.
se	Standard error of the coefficient.
f	An F statistic with one numerator degree of freedom.

Details

```
mp_assumptions(
  fixed_effects = list("(Intercept)" = 0, condition = mp_d_to_beta(0.5, sd = 1.12)),
   random_effects = list(subject = list(intercept_sd = mp_icc_to_sd(0.1, 1))),
   residual_sd = 1
)
```

- `mp_d_to_beta()` / `mp_beta_to_d()`: Cohen's d for a 0/1 predictor is the mean difference divided by `sd`, so the coefficient is `d * sd`.
- `mp_r2_to_beta()` / `mp_beta_to_r2()`: for a predictor with SD `predictor_sd`, a target (partial) variance-explained `r2` against residual SD `sd_resid` implies `beta = sqrt(r2/(1-r2)) * sd_resid / predictor_sd`.
- `mp_icc_to_sd()` / `mp_sd_to_icc()`: an intraclass correlation `icc` with residual SD `sd_resid` implies a random-intercept SD `sqrt(icc/(1-icc)) * sd_resid`.
- `mp_or_to_logodds()` / `mp_logodds_to_or()`: a binomial GLMM coefficient is the log odds ratio.

- `mp_t_to_beta()` / `mp_f_to_beta()`: recover a coefficient from a published t (or one-numerator-df F) statistic and its standard error.

Value

A numeric scalar.

Examples

```
mp_d_to_beta(0.5, sd = 1.12)
mp_icc_to_sd(0.1, sd_resid = 1)
mp_r2_to_beta(0.02, sd_resid = 1)
mp_or_to_logodds(1.5)
```

---

fit_model *Fit a model for a single simulated dataset*

Description

Fit a model for a single simulated dataset

Usage

```
fit_model(data, formula)
```

Arguments

`data`	A data.frame of simulated data.
`formula`	A model formula.

Value

A fitted model object.

---

mp_assumptions *Create modeling assumptions for simulation-based power*

Description

Assumptions encode effect sizes and the variance components used to simulate data. Random-effect sizes are given as standard deviations on the linear predictor (the scale lme4 reports), via `random_effects`.

Usage

```
mp_assumptions(
  fixed_effects,
  random_effects = NULL,
  icc = NULL,
  residual_sd = NULL,
  residual_ar1 = NULL,
  notes = NULL
)
```

Arguments

fixed_effects	Named list of numeric values (e.g., list("(Intercept)" = 0, condition = 0.4)).
random_effects	Optional named list keyed by grouping factor. Each element is a named list with intercept_sd (the random-intercept SD on the linear-predictor scale) and, optionally, slopes (a named list of random-slope SDs, one per predictor) and cor. For example, list(subject = list(intercept_sd = 0.5, slopes = list(condition = 0.3), cor = 0.2)) encodes a correlated random intercept and slope, i.e. (1 + condition | subject). With several slopes, cor may be a single scalar (applied to every pair of terms) or a full correlation matrix over c("(Intercept)", names(slopes)); each fixed effect named in fixed_effects also becomes a balanced design predictor.
icc	Deprecated. Previously documented as an intraclass correlation but used as a random-intercept SD. If supplied it is interpreted as intercept_sd and folded into random_effects with a warning. Use random_effects instead.
residual_sd	Optional non-negative numeric residual SD (Gaussian).
residual_ar1	Optional within-subject lag-1 autocorrelation of the Gaussian residuals, in (-1, 1), for longitudinal designs (residuals are drawn as a stationary AR(1) process ordered within each subject). Ignored for non-Gaussian families.
notes	Optional free text.

Value

An object of class mp_assumptions.

Examples

```
a <- mp_assumptions(
  fixed_effects = list("(Intercept)" = 0, condition = 0.4),
  random_effects = list(subject = list(intercept_sd = 0.5)),
  residual_sd = 1
)
a
```

mp_attach_calibration *Record a calibration result on a scenario*

Description

Attaches an mp_calibrate() result to a scenario so that mp_power() knows the Type I error has been checked (and will not emit its calibrate-first nudge). The calibration is stored on the scenario and travels with it.

Usage

```
mp_attach_calibration(scenario, calibration)
```

Arguments

scenario	An mp_scenario.
calibration	An mp_calibration object (from mp_calibrate()).

Value

The scenario with calibration attached.

See Also

mp_calibrate(), mp_plan().

mp_backend *MixPower backend contract*

Description

A **backend** is a list with three functions used by mp_power():

simulate_fun First argument receives the mp_scenario (often named scenario); optional seed. Must return a data.frame.

fit_fun Two arguments: simulated data.frame, then the mp_scenario (names may differ; mp_power() passes them positionally).

test_fun Two arguments: fitted model, then the mp_scenario; returns list(p_value = <numeric scalar>).

Usage

```
mp_backend(
  simulate_fun,
  fit_fun,
  test_fun,
  name = "custom",
  version = NULL,
  notes = NULL,
  capabilities = NULL
)
```

Arguments

simulate_fun	A data.frame-generating simulator (see Details).
fit_fun	Model fitter (see Details).
test_fun	Extracts a scalar p-value (see Details).
name	Short label for printing and manifests (default "custom").
version	Optional character version string for the backend implementation.
notes	Optional longer notes.
capabilities	Optional named list of flags (documentation only), e.g. list(families = c("gaussian"), supports_lrt = TRUE).

Details

Use mp_backend() to build a validated object of class mp_backend. Custom backends can also be plain lists with those three names; validate_mp_backend() checks the contract without requiring the class.

Value

An object of class c("mp_backend", "list") with the components above.

---

mp_backend_glmmtmb *Build a glmmTMB backend for Gaussian LMM scenarios*

Description

Fits with the **glmmTMB** function glmmTMB() (Gaussian). Useful for comparing simulation-based power to mp_backend_lme4() and for workflows that later extend to families supported by glmmTMB but not lme4.

Usage

```
mp_backend_glmmtmb(
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL,
  test_method = c("wald", "lrt"),
  null_formula = NULL
)
```

Arguments

predictor	Predictor column name.
subject	Subject ID column name.
outcome	Outcome column name.
item	Optional item ID column name.
test_method	Inference method: "wald" (normal-approximation z test, the fast default), "satterthwaite" or "kenward-roger" (df-corrected t tests via lmerTest/pbkrtest; recommended for small samples), "lrt" (likelihood-ratio test), or "pb" (parametric-bootstrap LRT via pbkrtest).
null_formula	Null-model formula required for "lrt" and "pb".

Value

An object of class mp_backend.

---

mp_backend_lme4 *Build an lme4 backend for MixPower scenarios*

Description

Build an lme4 backend for MixPower scenarios

Usage

```
mp_backend_lme4(
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL,
  test_method = c("wald", "lrt", "satterthwaite", "kenward-roger", "pb"),
  null_formula = NULL,
  pb_nsim = 100L
)
```

Arguments

predictor	Predictor column name.
subject	Subject ID column name.
outcome	Outcome column name.
item	Optional item ID column name.
test_method	Inference method: "wald" (normal-approximation z test, the fast default), "satterthwaite" or "kenward-roger" (df-corrected t tests via lmerTest/pbkrtest; recommended for small samples), "lrt" (likelihood-ratio test), or "pb" (parametric-bootstrap LRT via pbkrtest).
null_formula	Null-model formula required for "lrt" and "pb".
pb_nsim	Bootstrap replicates for test_method = "pb" (default 100). Note this multiplies cost: each power replicate refits the model pb_nsim times.

Value

A list containing simulate_fun, fit_fun, and test_fun.

---

mp_backend_lme4_binomial

Build an lme4 backend for binomial GLMM scenarios

Description

Build an lme4 backend for binomial GLMM scenarios

Usage

```
mp_backend_lme4_binomial(
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL,
  test_method = c("wald", "lrt", "pb"),
  null_formula = NULL,
  pb_nsim = 100L
)
```

Arguments

predictor	Predictor column name.
subject	Subject ID column name.
outcome	Outcome column name.
item	Optional item ID column name.

test_method	Inference method: "wald" (default), "lrt", or "pb" (parametric-bootstrap LRT via pbkrtest).
null_formula	Null-model formula required for "lrt" and "pb".
pb_nsim	Bootstrap replicates for test_method = "pb" (default 100).

Value

A list containing simulate_fun, fit_fun, and test_fun.

---

mp_backend_lme4_nb *Build an lme4 backend for Negative Binomial GLMM scenarios*

Description

Build an lme4 backend for Negative Binomial GLMM scenarios

Usage

```
mp_backend_lme4_nb(
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL,
  test_method = c("wald", "lrt", "pb"),
  null_formula = NULL,
  pb_nsim = 100L
)
```

Arguments

predictor	Predictor column name.
subject	Subject ID column name.
outcome	Outcome column name.
item	Optional item ID column name.
test_method	Inference method: "wald" (default), "lrt", or "pb" (parametric-bootstrap LRT via pbkrtest).
null_formula	Null-model formula required for "lrt" and "pb".
pb_nsim	Bootstrap replicates for test_method = "pb" (default 100).

Value

A list containing simulate_fun, fit_fun, and test_fun.

mp_backend_lme4_poisson *Build an lme4 backend for Poisson GLMM scenarios*

Description

Build an lme4 backend for Poisson GLMM scenarios

Usage

```
mp_backend_lme4_poisson(
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL,
  test_method = c("wald", "lrt", "pb"),
  null_formula = NULL,
  pb_nsim = 100L
)
```

Arguments

predictor	Predictor column name.
subject	Subject ID column name.
outcome	Outcome column name.
item	Optional item ID column name.
test_method	Inference method: "wald" (default), "lrt", or "pb" (parametric-bootstrap LRT via pbkrtest).
null_formula	Null-model formula required for "lrt" and "pb".
pb_nsim	Bootstrap replicates for test_method = "pb" (default 100).

Value

A list containing simulate_fun, fit_fun, and test_fun.

mp_bundle_results *Bundle results with manifest and optional labels*

Description

Combines a single result object (mp_power, mp_sensitivity, or mp_power_curve), a reproducibility manifest, and optional user labels into one object. Diagnostics and result structure are retained unchanged.

Usage

```
mp_bundle_results(
  result,
  manifest,
  study_id = NULL,
  analyst = NULL,
  notes = NULL
)
```

Arguments

result	An object of class mp_power, mp_sensitivity, or mp_power_curve.
manifest	An mp_manifest object (from mp_manifest()).
study_id	Optional character; study or run identifier.
analyst	Optional character; analyst name or ID.
notes	Optional character; free-form notes.

Value

An object of class mp_bundle with components result, manifest, and labels (list with study_id, analyst, notes).

---

mp_calibrate *Check the Type I error calibration of a scenario's test*

Description

Runs the scenario under the null hypothesis — the focal fixed effect set to zero — and estimates the empirical Type I error rate, i.e. the proportion of replicates in which the (false) effect is declared significant. A trustworthy test rejects at approximately alpha. The estimate is compared to alpha using an exact (Clopper-Pearson) interval, giving a verdict:

Usage

```
mp_calibrate(
  scenario,
  term = NULL,
  nsim = 1000,
  alpha = 0.05,
  seed = NULL,
  conf_level = 0.95,
  failure_policy = c("count_as_nondetect", "exclude")
)
```

Arguments

scenario	An mp_scenario.
term	Focal fixed-effect term to null out. Defaults to the scenario's test term (or the first non-intercept fixed effect).
nsim	Number of null simulations (default 1000; Type I estimation needs more replicates than a power point estimate for a tight interval).
alpha	Nominal significance level being checked (default 0.05).
seed	Optional seed for reproducibility.
conf_level	Confidence level for the Type I interval (default 0.95).
failure_policy	Passed to mp_power().

Details

- "well-calibrated": the interval contains alpha.
- "anti-conservative": the interval lies entirely above alpha (the test rejects too often — e.g. a Wald test with few clusters, or a model that omits a random slope that is actually present in the data-generating process). Power computed with such a test is not trustworthy.
- "conservative": the interval lies entirely below alpha.

This is the recommended sanity check to run before trusting a power number: if a design/analysis combination does not control Type I error, its power is meaningless.

Value

An object of class mp_calibration: a list with term, alpha, type1 (empirical Type I rate), ci, conf_level, mcse, nsim, verdict, and the underlying power_result.

See Also

mp_recommend_method(), mp_power().

Examples

```
if (requireNamespace("lme4", quietly = TRUE)) {
  d <- mp_design(list(subject = 40), trials_per_cell = 6)
  a <- mp_assumptions(
    fixed_effects = list("(Intercept)" = 0, condition = 0.4),
    random_effects = list(subject = list(intercept_sd = 0.5)),
    residual_sd = 1
  )
  scn <- mp_scenario_lme4(y ~ condition + (1 | subject), design = d, assumptions = a)
  mp_calibrate(scn, nsim = 200, seed = 1)
}
```

`mp_compare_models` *Compare analysis models on the same data: power* and *Type I error*

Description

Simulates data once per replicate from the first scenario, then fits and tests *every* supplied scenario on that same dataset. Because the models see identical data, differences in their rejection rates isolate the effect of the analysis choice. This is the analogue of powerlmm's `sim_formula`, raised to a planning question: **which analysis plan controls Type I error AND has acceptable power under your plausible data-generating process?**

Usage

```
mp_compare_models(
  scenarios,
  nsim,
  alpha = 0.05,
  seed = NULL,
  conf_level = 0.95,
  failure_policy = c("count_as_nondetect", "exclude"),
  calibrate = TRUE
)
```

Arguments

`scenarios`	A named list of `mp_scenario` objects.
`nsim`	Positive integer number of simulations (per pass).
`alpha`	Significance threshold (default 0.05).
`seed`	Optional seed for reproducibility.
`conf_level`	Confidence level for the per-model intervals.
`failure_policy`	How to treat failed fits (see `mp_power()`).
`calibrate`	Also estimate each model's Type I error under the null (default `TRUE`); doubles the simulation cost.

Details

With `calibrate = TRUE` (default), each candidate model is evaluated twice on shared data: once under the scenario's alternative (giving **power**) and once under the null — the focal effect set to zero in the data-generating scenario — giving its **Type I error rate** and a calibration verdict. The recommended plan is the best-powered model that still controls Type I error.

All scenarios must share the same data-generating process (design and assumptions); they should differ only in their analysis model (formula / random-effects structure / test). The first scenario's `simulate_fun` drives data generation.

Value

An object of class mp_model_comparison with a results data frame (one row per model: power, conf_low, conf_high; and, when calibrate, type1, type1_low, type1_high, calibration, valid; plus failure_rate, n_effective, nsim) and $recommended, the best-powered Type-I-valid model.

Examples

```
if (requireNamespace("lme4", quietly = TRUE)) {
  d <- mp_design(list(subject = 30), trials_per_cell = 8)
  a <- mp_assumptions(
    fixed_effects = list("(Intercept)" = 0, condition = 0.5),
    random_effects = list(subject = list(intercept_sd = 0.5,
                                         slopes = list(condition = 0.8))),
    residual_sd = 1
  )
  maximal <- mp_scenario_lme4(y ~ condition + (1 + condition | subject), d, a)
  reduced <- mp_scenario_lme4(y ~ condition + (1 | subject), d, a)
  mp_compare_models(list(maximal = maximal, reduced = reduced),
                    nsim = 100, seed = 1)
}
```

---

mp_custom_dgp *Plug a custom data-generating function into the mixpower pipeline*

Description

Wraps a user-supplied data-generating function as a validated simulate_fun, so any design the built-in grammar does not cover — partially nested designs, multi-arm factors, nonlinear time trends, correlated predictors, or anything else — can still flow through mp_power(), mp_calibrate(), mp_compare_models(), and the reporting tools. The returned function is the escape hatch that keeps the core engine lean while leaving the design grammar open-ended.

Usage

```
mp_custom_dgp(fun, validate = TRUE)
```

Arguments

fun	A function function(scenario, seed = NULL) returning a data.frame.
validate	Validate the output each call (default TRUE): a data frame that contains all variables in scenario$formula.

Details

The user function receives the scenario (and optionally `seed`) and must return a `data.frame` containing every variable in the scenario's model formula (the response and predictors/grouping factors). To keep `mp_calibrate()` / `mp_compare_models()` meaningful, read the focal effect from `scenario$assumptions$fixed_effects` rather than hard-coding it, so the null (effect = 0) can be generated when those tools set it.

Value

A function suitable as a scenario `simulate_fun` (see `mp_scenario()` and the `simulate` argument of `mp_scenario_lme4()`).

See Also

`mp_scenario_lme4()`, `mp_scenario()`.

Examples

```
if (requireNamespace("lme4", quietly = TRUE)) {
  # A three-arm design (omnibus test) via a custom DGP.
  dgp <- function(scenario, seed = NULL) {
    n <- scenario$design$clusters$subject
    b <- scenario$assumptions$fixed_effects
    arm <- factor(rep(c("a", "b", "c"), length.out = n))
    mu <- c(a = 0, b = b$arm_b, c = b$arm_c)[as.character(arm)]
    data.frame(subject = factor(seq_len(n)), arm = arm,
               y = mu + stats::rnorm(n, sd = scenario$assumptions$residual_sd))
  }
  d <- mp_design(list(subject = 90), trials_per_cell = 1)
  a <- mp_assumptions(list("(Intercept)" = 0, arm_b = 0.4, arm_c = 0.7),
                      residual_sd = 1)
  scn <- mp_scenario_lme4(y ~ arm + (1 | subject), design = d, assumptions = a,
                          simulate = dgp, test_method = "lrt",
                          null_formula = y ~ 1 + (1 | subject))
  mp_power(scn, nsim = 20, seed = 1)
}
```

`mp_design` *Create a study design specification*

Description

`mp_design()` encodes how data will be collected: cluster sizes and repeated measurements. It does not encode effect sizes or analysis decisions.

Usage

```
mp_design(
  clusters,
  trials_per_cell = 1,
  predictors = NULL,
  nesting = NULL,
  notes = NULL
)
```

Arguments

clusters	Named list of positive integers. Example: `list(subject = 50, item = 30)`. For a nested (three-level) design, a nested factor's count is interpreted as the number of units *per parent* (see `nesting`).
trials_per_cell	Number of repeated observations per subject. A single positive integer (balanced), or a positive-integer vector recycled across subjects for an unbalanced design.
predictors	Optional named list giving the design type of each predictor (each non-intercept fixed effect). Each entry is either a string (`"binary"` or `"continuous"`) or a list with `type`, `level`, and an optional `allocation`. `level` is `"within"` (varies within subject), `"between"` (constant within subject, varies across subjects), or `"cluster"` (assigned at the nesting parent, i.e. cluster-randomized; requires `nesting`). `allocation` (a proportion in `(0, 1)`) sets an unequal split for a binary between/cluster predictor (default balanced). Unspecified predictors default to a balanced within-subject binary factor. Example: `list(time = list(type = "continuous", level = "within"), arm = list(level = "cluster", allocation = 0.5))`.
nesting	Optional named character vector mapping a child grouping factor to its parent, e.g. `c(subject = "site")` for subjects nested in sites. The parent must also appear in `clusters`.
notes	Optional free text.

Value

An object of class `mp_design`.

Examples

```
d <- mp_design(clusters = list(subject = 40), trials_per_cell = 10)
d

# Three-level: 8 sites, 5 subjects per site, 4 trials each.
d3 <- mp_design(
  clusters = list(site = 8, subject = 5),
  trials_per_cell = 4,
  nesting = c(subject = "site")
)
```

`mp_elicit` *Elicit assumptions from standardized scientific quantities*

Description

Build a coherent `mp_assumptions()` object from the quantities researchers actually reason about — intraclass correlations, a standardized effect, a baseline event probability and an odds/risk ratio, and random-slope variability — instead of hand-picking raw regression coefficients and standard deviations. `mp_elicit()` solves for an internally consistent parameter set and prints the **implied model**, combining mlmpower-style elicitation with mixpower's simulation and calibration tooling.

Usage

```
mp_elicit(
  family = c("gaussian", "binomial"),
  predictor = "condition",
  d = NULL,
  odds_ratio = NULL,
  risk_ratio = NULL,
  baseline_prob = NULL,
  outcome_sd = 1,
  icc = NULL,
  slope = NULL,
  cor = NULL,
  groups = "subject",
  intercept = NULL
)
```

Arguments

`family`	`"gaussian"` or `"binomial"`.
`predictor`	Name of the focal predictor (default `"condition"`).
`d`	Standardized effect (Cohen's d) for `"gaussian"`.
`odds_ratio`, `risk_ratio`	Effect for `"binomial"` (give one).
`baseline_prob`	Control-group event probability for `"binomial"`.
`outcome_sd`	Total outcome SD for `"gaussian"` (default 1).
`icc`	Intraclass correlation: a single value (applied to all `groups`) or a vector named by group. For `"gaussian"` the values must sum to `< 1`.
`slope`	Random-slope SD as a fraction of the random-intercept SD: a single value or named by group (`NULL` for none).
`cor`	Intercept-slope correlation: a single value or named by group.
`groups`	Grouping-factor name(s) (default `"subject"`).
`intercept`	Fixed intercept; defaults to 0 (gaussian) or `qlogis(baseline_prob)` (binomial).

Details

For `family = "gaussian"` the total outcome variance is partitioned by the per-group `icc` (each random-intercept variance is `icc * outcome_sd^2`, the residual variance is the remainder), and the fixed effect is `d * outcome_sd`. For `family = "binomial"` the intercept is `qlogis(baseline_prob)`, the effect is `log(odds_ratio)` (a `risk_ratio` is converted given `baseline_prob`), and each random-intercept SD is derived from the latent-scale `icc` (`residual variance = pi^2 / 3`). A random slope, when requested, is set to `slope` times the random-intercept SD.

Predictor correlations, autocorrelation, and cluster-size distributions are design choices (see `mp_design()`); missingness is `mp_missing()`.

Value

An object of class `c("mp_elicitation", "mp_assumptions")` usable anywhere `mp_assumptions()` is; its `print()` shows the elicited inputs and the implied model.

See Also

`mp_assumptions()`, `mp_d_to_beta()`, `mp_icc_to_sd()`.

Examples

```
# Gaussian: medium effect, ICC = 0.1
mp_elicit("gaussian", d = 0.5, icc = 0.1)

# Binomial: 20% baseline event rate, odds ratio 1.8, subject ICC 0.05
mp_elicit("binomial", baseline_prob = 0.2, odds_ratio = 1.8, icc = 0.05)
```

---

mp_extend *Scale a fitted-model scenario's sample size up or down*

Description

Sets target level counts for one or more grouping factors of a scenario built with `mp_from_fit()`, so power can be evaluated at a sample size different from the pilot's. This is the analogue of `simr::extend()`: levels are cloned from the pilot's within-level structure with fresh ids, and fresh random effects are drawn from the fitted covariance, so the extended data represent a larger (or smaller) sample from the same population.

Usage

```
mp_extend(scenario, ...)
```

Arguments

`scenario`	An `mp_scenario` created by `mp_from_fit()`.
`...`	Named target level counts, e.g. `Subject = 60`.

Details

Pair with `mp_power()` for a single N, or with `mp_power_curve()` / `mp_solve_sample_size()` using the `extend.<group>` key for a curve over N.

Value

The scenario with its `extend` targets set.

See Also

`mp_from_fit()`.

Examples

```
if (requireNamespace("lme4", quietly = TRUE)) {
  m <- lme4::lmer(Reaction ~ Days + (Days | Subject), data = lme4::sleepstudy)
  scn <- mp_from_fit(m, test_term = "Days")
  big <- mp_extend(scn, Subject = 40)
  mp_power(big, nsim = 20, seed = 1)
}
```

`mp_from_fit` *Build a power scenario from a fitted lme4 model*

Description

Turns an existing `lmer`/`glmer` fit (e.g. from pilot or published data) into an `mp_scenario`, so its estimated effects and variance components inform a simulation-based power analysis. New responses are simulated from the fitted model with `stats::simulate()` (keeping the estimated random-effect structure and residual variance), the model is refit, and the focal term is tested.

Usage

```
mp_from_fit(
  fit,
  test_term = NULL,
  test_method = NULL,
  null_formula = NULL,
  pb_nsim = 100L,
  extend = NULL
)
```

Arguments

fit	A fitted model of class lmerMod/lmerModLmerTest (Gaussian LMM) or glmerMod (binomial/Poisson/negative-binomial GLMM).
test_term	Fixed-effect term to test. Defaults to the first non-intercept fixed effect.
test_method	Inference method. Gaussian fits allow "wald" (default), "satterthwaite", "kenward-roger", "lrt", "pb"; GLMM fits allow "wald", "lrt", "pb".
null_formula	Null-model formula required for "lrt"/"pb". Defaults to the fitted formula with test_term removed.
pb_nsim	Bootstrap replicates for test_method = "pb" (default 100).
extend	Optional named list of target level counts per grouping factor (e.g. list(Subject = 60)) used to scale the pilot's sample size up or down. Levels are cloned from the pilot's structure with fresh ids and fresh random effects drawn from the fitted covariance. See mp_extend() and the extend.<group> sensitivity key for power curves over N.

Details

The fixed effects used to simulate are read from the scenario's assumptions, which start at the fitted coefficients. This means mp_sensitivity() / mp_power_curve() can vary an effect size (e.g. fixed_effects.condition) for data-based vs smallest-effect-of-interest comparisons. Sample size can be scaled up or down from the pilot with mp_extend() (or the extend.<group> sensitivity key), which clones the pilot's structure with fresh levels and fresh random effects.

Value

An object of class mp_scenario.

Examples

```
if (requireNamespace("lme4", quietly = TRUE)) {
  m <- lme4::lmer(Reaction ~ Days + (Days | Subject), data = lme4::sleepstudy)
  scn <- mp_from_fit(m, test_term = "Days")
  mp_power(scn, nsim = 20, seed = 1)
}
```

---

mp_grid_sample_size *Create a grid of values for sample-size search*

Description

Returns a numeric vector suitable for mp_solve_sample_size()'s grid argument. Specify either the number of points (length.out) or the step size (by); bounds are always explicit (from, to).

Usage

```
mp_grid_sample_size(from, to, length.out = NULL, by = NULL)
```

Arguments

from	Lower bound (inclusive).
to	Upper bound (inclusive).
length.out	Number of points (optional). Uses seq() with length.out; from and to are the first and last values.
by	Step size (optional). Uses seq() with by; sequence runs from from to to in steps of by.

Value

A numeric vector. For integer cluster sizes, use round(mp_grid_sample_size(...)) or pass by as an integer (e.g. by = 10).

Examples

```
mp_grid_sample_size(20, 100, length.out = 9)
mp_grid_sample_size(20, 100, by = 10)
```

---

mp_manifest *Reproducibility manifest for power analyses*

Description

Captures scenario fingerprint, seed strategy, session info, timestamp, and optional git SHA so results can be reproduced or audited. Output is a plain list (and one-row data frame via as.data.frame()) suitable for saving alongside results.

Usage

```
mp_manifest(scenario, seed = NULL, session = TRUE)
```

Arguments

scenario	An mp_scenario object (used for digest).
seed	The seed value used (or NULL). Stored as-is; strategy is inferred as "fixed" if non-null else "none".
session	Include full sessionInfo() (default TRUE). If FALSE, only R version and mixpower version are stored.

Value

A list with components: scenario_digest, seed, seed_strategy, timestamp, r_version, mixpower_version, session_info (if requested), git_sha (if in a git repo). Use as.data.frame() on the list for a single-row table (list components become columns where possible).

mp_methods_text *Generate a methods paragraph for a power analysis*

Description

Produces a ready-to-edit prose description of a simulation-based power analysis from an `mp_power()` result: the design, model, effect tested, inference method, number of simulations, and the estimated power with its interval. Intended to seed the "Power analysis" paragraph of a methods section.

Usage

```
mp_methods_text(result, software = TRUE)
```

Arguments

`result`	An `mp_power` object (from `mp_power()`).
`software`	Include a sentence naming the mixpower package and version (default `TRUE`).

Value

A length-1 character string with class `mp_methods_text` (its `print()` method word-wraps the paragraph).

Examples

```
if (requireNamespace("lme4", quietly = TRUE)) {
  d <- mp_design(list(subject = 30), trials_per_cell = 6)
  a <- mp_assumptions(list("(Intercept)" = 0, condition = 0.4),
                      random_effects = list(subject = list(intercept_sd = 0.5)),
                      residual_sd = 1)
  scn <- mp_scenario_lme4(y ~ condition + (1 | subject), design = d, assumptions = a)
  mp_methods_text(mp_power(scn, nsim = 50, seed = 1))
}
```

mp_missing *Add a missing-data / dropout mechanism to a scenario*

Description

Wraps a scenario so that, on every replicate, observations are deleted from the simulated data before the model is fit. This lets a power analysis reflect realistic incomplete data (Gallop & Liu, 2017; Magnusson, 2018). Three mechanisms are supported:

Usage

```
mp_missing(
  scenario,
  mechanism = c("mcar", "mar", "dropout"),
  prob = NULL,
  on = NULL,
  slope = 0,
  time = NULL,
  dropout = NULL,
  subject = "subject"
)
```

Arguments

scenario	An mp_scenario (any backend).
mechanism	One of "mcar", "mar", "dropout".
prob	Baseline deletion probability for "mcar"/"mar".
on	Name of the observed column the "mar" probability depends on.
slope	Logit-scale slope for "mar" (default 0).
time	Name of the within-subject ordering column for "dropout".
dropout	For "dropout": a numeric vector of cumulative dropout proportions per ordered timepoint, or list(shape=, scale=) for Weibull.
subject	Subject grouping column (default "subject").

Details

- "mcar": each observation is deleted independently with probability prob (missing completely at random).
- "mar": each observation is deleted with a probability that depends on an *observed* column on through a logistic model plogis(qlogis(prob) + slope * on) (missing at random).
- "dropout": monotone longitudinal dropout along time within each subject — once a subject drops out it contributes no later observations. The dropout pattern is given either by dropout as a vector of cumulative dropout proportions (one per ordered timepoint) or as list(shape =, scale =) for a Weibull dropout time on the time scale.

Value

The scenario with its simulator wrapped to apply missingness.

Examples

```
if (requireNamespace("lme4", quietly = TRUE)) {
  d <- mp_design(list(subject = 30), trials_per_cell = 6,
                 predictors = list(time = "continuous"))
  a <- mp_assumptions(list("(Intercept)" = 0, time = 0.4),
                      random_effects = list(subject = list(intercept_sd = 0.5)),
```

```
                      residual_sd = 1)
  scn <- mp_scenario_lme4(y ~ time + (1 | subject), design = d,
                          assumptions = a, predictor = "time")
  scn_drop <- mp_missing(scn, "dropout", time = "time",
                         dropout = c(0, 0.1, 0.2, 0.35, 0.5, 0.6))
  mp_power(scn_drop, nsim = 20, seed = 1)
}
```

`mp_plan` *Plan power the calibrate-first way*

Description

The recommended end-to-end workflow: **calibrate the null, choose an inference method, then estimate power, and report both**. `mp_plan()` runs `mp_calibrate()` and `mp_power()` on the same scenario and bundles them with an `mp_recommend_method()` suggestion, so power is never reported without the Type I error check that tells you whether it is trustworthy.

Usage

```
mp_plan(
  scenario,
  nsim = 1000,
  calibrate_nsim = nsim,
  alpha = 0.05,
  seed = NULL,
  conf_level = 0.95,
  ci_method = c("clopper-pearson", "wald"),
  failure_policy = c("count_as_nondetect", "exclude")
)
```

Arguments

`scenario`	An `mp_scenario`.
`nsim`	Simulations for the power estimate (default 1000).
`calibrate_nsim`	Simulations for the null calibration (default `nsim`).
`alpha`	Significance level (default 0.05).
`seed`	Optional seed for reproducibility.
`conf_level`	Confidence level for intervals (default 0.95).
`ci_method`	Power CI type (see `mp_power()`).
`failure_policy`	Failure policy (see `mp_power()`).

Value

An object of class `mp_plan`: a list with `calibration`, `power`, `recommendation`, `scenario`, and `alpha`.

See Also

mp_calibrate(), mp_power(), mp_recommend_method().

Examples

```
if (requireNamespace("lme4", quietly = TRUE)) {
  d <- mp_design(list(subject = 12), trials_per_cell = 8)
  a <- mp_assumptions(
    fixed_effects = list("(Intercept)" = 0, condition = 0.4),
    random_effects = list(subject = list(intercept_sd = 0.5)),
    residual_sd = 1
  )
  scn <- mp_scenario_lme4(y ~ condition + (1 | subject), design = d, assumptions = a)
  mp_plan(scn, nsim = 100, seed = 1)
}
```

---

mp_power *Simulation-based power estimation (engine-agnostic core)*

Description

mp_power() runs repeated simulations under a scenario and estimates power for the scenario's test decision rule (typically p < alpha).

Usage

```
mp_power(
  scenario,
  nsim,
  alpha = 0.05,
  seed = NULL,
  failure_policy = c("count_as_nondetect", "exclude"),
  keep = c("minimal", "fits", "data"),
  conf_level = 0.95,
  ci_method = c("clopper-pearson", "wald"),
  aggregate = c("full", "streaming"),
  check_calibration = TRUE
)
```

Arguments

scenario	An mp_scenario.
nsim	Positive integer number of simulations.
alpha	Significance threshold for a detection (default 0.05).
seed	Optional seed for reproducibility.
failure_policy	How to treat failed fits / missing p-values:

- "count_as_nondetect" (default): failures count as non-detections.
- "exclude": drop failures from the denominator (always reported).

keep — What to store:

- "minimal": only per-sim summary rows.
- "fits": also store fit objects (may be large).
- "data": also store simulated data (can be very large).

conf_level — Confidence level for the power interval (default 0.95).

ci_method — Power CI type: "clopper-pearson" (default, exact binomial) or "wald" (normal approximation).

aggregate — "full" (default) stores every replicate in sims. "streaming" accumulates counts only (lower memory); requires keep = "minimal" and returns an empty sims data frame with the same column names as the full run.

check_calibration — If TRUE (default), warn once per session when the design is risky for Type I error (few clusters and/or complex random effects) and no calibration is on record, nudging toward mp_calibrate() / mp_plan(). The advice is always available as $calibration_advice on the result. Set FALSE to silence (used internally by parameter sweeps).

Details

In Phase 4 core, the scenario must provide engine functions: simulate_fun, fit_fun, and test_fun. Later phases will supply defaults based on specific backends (e.g., lme4).

Diagnostics include Type S (wrong-sign) and Type M (exaggeration-ratio) errors among significant replicates (Gelman & Carlin, 2014), computed when the tested term's true effect is known and non-zero and the backend reports an estimate.

Value

An object of class mp_power.

Examples

```
# A tiny toy engine (not mixed models) just to demonstrate the workflow:
d <- mp_design(list(subject = 30), trials_per_cell = 1)
a <- mp_assumptions(list(condition = 0.3), residual_sd = 1)

sim_fun <- function(scn, seed) {
  n <- scn$design$clusters$subject
  x <- stats::rbinom(n, 1, 0.5)
  y <- scn$assumptions$fixed_effects$condition * x +
    stats::rnorm(n, sd = scn$assumptions$residual_sd)
  data.frame(y = y, condition = x)
}
fit_fun <- function(dat, scn) stats::lm(scn$formula, data = dat)
test_fun <- function(fit, scn) {
  sm <- summary(fit)
  p <- sm$coefficients["condition", "Pr(>|t|)"]
```

```
  list(p_value = as.numeric(p))
}

s <- mp_scenario(
  y ~ condition, d, a,
  simulate_fun = sim_fun,
  fit_fun = fit_fun,
  test_fun = test_fun
)
res <- mp_power(s, nsim = 50, seed = 1)
summary(res)
```

`mp_power_adaptive` *Adaptive (sequential) power simulation*

Description

Runs `mp_power()` in batches and stops as soon as the result is precise enough or the accept/reject decision is clear (see `mp_stop()`), instead of relying on an arbitrary fixed `nsim`. The achieved Monte Carlo precision (CI half-width) and the number of replicates actually used are reported, so you simulate no more than necessary.

Usage

```
mp_power_adaptive(
  scenario,
  stop = mp_stop(),
  alpha = 0.05,
  seed = NULL,
  conf_level = 0.95,
  ci_method = c("clopper-pearson", "wald"),
  failure_policy = c("count_as_nondetect", "exclude"),
  check_calibration = TRUE,
  progress = FALSE
)
```

Arguments

`scenario`	An `mp_scenario`.
`stop`	An `mp_stop()` specification.
`alpha`, `seed`, `conf_level`, `ci_method`, `failure_policy`	As in `mp_power()`.
`check_calibration`	As in `mp_power()` (calibrate-first nudge); checked once at the start.
`progress`	Emit a per-batch progress message (default `FALSE`).

Details

Because replicate seeds are deterministic (`seed + i - 1`), the result for a given stopping point is reproducible and identical to a fixed `mp_power()` run at the same `nsim`.

Value

An object of class `c("mp_power_adaptive", "mp_power")` — a standard `mp_power` result plus a `$stopping` list (`converged`, `reason`, `nsim_used`, `ci_halfwidth`, `target`, `rule`).

See Also

`mp_stop()`, `mp_power()`, `mp_power_checkpoint()`.

Examples

```
if (requireNamespace("lme4", quietly = TRUE)) {
  d <- mp_design(list(subject = 30), trials_per_cell = 6)
  a <- mp_assumptions(list("(Intercept)" = 0, condition = 0.5),
                      random_effects = list(subject = list(intercept_sd = 0.5)),
                      residual_sd = 1)
  scn <- mp_scenario_lme4(y ~ condition + (1 | subject), design = d, assumptions = a)
  mp_power_adaptive(scn, stop = mp_stop(ci_halfwidth = 0.05, batch = 50,
                                        min_nsim = 50, max_nsim = 400), seed = 1)
}
```

`mp_power_checkpoint` *Resumable, checkpointed power simulation*

Description

Runs `mp_power()` in batches, saving the accumulated per-replicate results to `file` (an `.rds`) after each batch. If `file` already exists for the same `seed`, the run resumes from where it left off and only the remaining replicates are simulated. This makes very large or long-running power analyses robust to interruption, and lets you grow `nsim` later without recomputing finished replicates.

Usage

```
mp_power_checkpoint(
  scenario,
  nsim,
  file,
  batch_size = 100,
  alpha = 0.05,
  seed = NULL,
  failure_policy = c("count_as_nondetect", "exclude"),
  conf_level = 0.95,
  ci_method = c("clopper-pearson", "wald"),
```

```
  progress = TRUE
)
```

Arguments

scenario	An mp_scenario.
nsim	Total number of simulations desired.
file	Path to the checkpoint .rds file.
batch_size	Replicates per batch (default 100).
alpha, seed, failure_policy, conf_level, ci_method	As in mp_power(). seed must be non-NULL for reproducible, resumable batches.
progress	Emit a progress message after each batch (default TRUE).

Details

Because replicate seeds are deterministic (seed + i - 1), the result is identical to a single mp_power(scenario, nsim, seed = seed) call: batching only changes when work happens, not what is computed.

Value

An object of class mp_power for all nsim replicates.

See Also

mp_power().

Examples

```
if (requireNamespace("lme4", quietly = TRUE)) {
  d <- mp_design(list(subject = 30), trials_per_cell = 6)
  a <- mp_assumptions(list("(Intercept)" = 0, condition = 0.4),
                      random_effects = list(subject = list(intercept_sd = 0.5)),
                      residual_sd = 1)
  scn <- mp_scenario_lme4(y ~ condition + (1 | subject), design = d, assumptions = a)
  f <- tempfile(fileext = ".rds")
  mp_power_checkpoint(scn, nsim = 60, file = f, batch_size = 20, seed = 1)
}
```

---

mp_power_curve *Power curve for a single design/assumption parameter*

Description

Runs mp_power() across a one-dimensional grid of values for one parameter (e.g. cluster size) via mp_sensitivity(). Results include power estimates and per-grid-point diagnostics: failure rate, singular rate, and effective N.

Usage

```
mp_power_curve(
  scenario,
  vary,
  nsim = 100,
  alpha = 0.05,
  seed = NULL,
  failure_policy = c("count_as_nondetect", "exclude"),
  conf_level = 0.95
)
```

Arguments

scenario	An mp_scenario.
vary	Named list with a single key (e.g. clusters.subject).
nsim	Number of simulations per grid point (default 100).
alpha	Significance level (default 0.05).
seed	Optional seed for reproducibility.
failure_policy	How to treat failed fits: "count_as_nondetect" or "exclude".
conf_level	Confidence level for power intervals (default 0.95).

Value

An object of class mp_power_curve with components vary, grid, results (estimate, mcse, conf_low, conf_high, failure_rate, singular_rate, n_effective, nsim, plus the varying parameter column), alpha, failure_policy, and conf_level.

---

mp_power_curve_parallel

Parallel power curve evaluation

Description

Evaluates power over a one-parameter grid by running mp_power() for each grid cell in parallel. Uses explicit per-cell seeds (seed + cell_index - 1L) so results are deterministic and match serial mp_power_curve() for the same seed. Does not modify mp_power(); parallelization is at the scenario-grid level only.

Usage

```
mp_power_curve_parallel(
  scenario,
  vary,
  workers = 2L,
  nsim = 100,
```

```
  alpha = 0.05,
  seed = NULL,
  failure_policy = c("count_as_nondetect", "exclude"),
  conf_level = 0.95,
  progress = FALSE,
  ...
)
```

Arguments

scenario	An mp_scenario.
vary	Named list with exactly one parameter (e.g. clusters.subject).
workers	Number of parallel workers (default 2).
nsim	Number of simulations per grid point (default 100).
alpha	Significance level (default 0.05).
seed	Optional base seed; each cell gets seed + cell_index - 1L.
failure_policy	How to treat failed fits: "count_as_nondetect" or "exclude".
conf_level	Confidence level for power intervals (default 0.95).
progress	If TRUE, run serially with a progress bar; if FALSE, run in parallel.
...	Unused; reserved for future arguments.

Value

An object of class mp_power_curve (same structure as mp_power_curve()).

Note

Parallel execution requires the **parallel** package (base R) and that **mixpower** is installed (e.g. install.packages() or devtools::install()) so that workers can load it.

---

mp_quick_power *Quick power run for a single LMM design*

Description

One-call wrapper that builds design, assumptions, scenario (lme4 LMM), and runs mp_power(). Intended for the common case: one fixed effect, one random intercept. All arguments are explicit; pass further options to mp_power() via ... (e.g. failure_policy, conf_level, keep).

Usage

```
mp_quick_power(
  formula,
  clusters,
  trials_per_cell = 1,
  fixed_effects,
  residual_sd,
  nsim,
  alpha = 0.05,
  seed = NULL,
  random_effects = NULL,
  icc = NULL,
  test_method = c("wald", "lrt"),
  null_formula = NULL,
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL,
  ...
)
```

Arguments

formula	Model formula (e.g. y ~ condition + (1 | subject)).
clusters	Named list of cluster sizes (e.g. list(subject = 40)).
trials_per_cell	Number of observations per cell (default 1).
fixed_effects	Named list of effect sizes (e.g. list(condition = 0.3)). Include intercept as (Intercept) if needed.
residual_sd	Residual standard deviation.
nsim	Number of simulations.
alpha	Significance level (default 0.05).
seed	Optional seed for reproducibility.
random_effects	Optional named list of random-effect sizes, e.g. list(subject = list(intercept_sd = 0.5)). See mp_assumptions().
icc	Deprecated; interpreted as a random-intercept SD. Use random_effects.
test_method	"wald" (default) or "lrt".
null_formula	Required when test_method = "lrt" (e.g. y ~ 1 + (1 | subject)).
predictor	Predictor column name (default "condition").
subject	Subject ID column name (default "subject").
outcome	Outcome column name (default "y").
item	Optional item ID column name.
...	Arguments passed to mp_power() (e.g. failure_policy, conf_level, keep).

Value

The result of `mp_power()` (object of class `mp_power`).

Examples

```
mp_quick_power(
  y ~ condition + (1 | subject),
  clusters = list(subject = 40),
  trials_per_cell = 8,
  fixed_effects = list(`(Intercept)` = 0, condition = 0.3),
  residual_sd = 1,
  nsim = 50,
  seed = 123
)
```

`mp_recommend_method` *Recommend an inference method for a scenario*

Description

Heuristic guidance on the `test_method` for a scenario, based on the number of levels of the random grouping factors. Wald (normal-approximation z/t) tests and, to a lesser extent, likelihood-ratio tests are known to be anti-conservative when the number of clusters is small (Luke, 2017): the degrees of freedom are overstated, so the test rejects too often. With few clusters, a degrees-of-freedom-corrected test (Satterthwaite or Kenward-Roger, for linear mixed models) or a parametric bootstrap (any family) controls Type I error far better.

Usage

```
mp_recommend_method(scenario, small_clusters = 30L)
```

Arguments

`scenario`	An `mp_scenario`.
`small_clusters`	Threshold below which the smallest grouping factor is treated as "few clusters" (default 30).

Details

This is a fast, design-based heuristic; to *measure* a specific design and method, use `mp_calibrate()`.

Value

An object of class `mp_recommendation`: a list with `method` (the scenario's current method), `n_groups` (smallest grouping-factor size), `is_lmm`, `caution` (logical), `recommended` (character vector), and `rationale`.

See Also

mp_calibrate().

Examples

```
d <- mp_design(list(subject = 12), trials_per_cell = 8)
a <- mp_assumptions(
  fixed_effects = list("(Intercept)" = 0, condition = 0.4),
  random_effects = list(subject = list(intercept_sd = 0.5)),
  residual_sd = 1
)
scn <- mp_scenario_lme4(y ~ condition + (1 | subject), design = d, assumptions = a)
mp_recommend_method(scn)
```

---

mp_report *Build a full "trust report" for a power analysis*

Description

Assembles a single, reviewer-ready report from a power result: the design, model, data-generating assumptions, inference method, Type I calibration, power estimate with interval and Monte Carlo precision, diagnostics (failure/singular-fit rates, Type S/M), missing-data assumptions, reproducibility manifest (software versions, seed), and — importantly — explicit warnings about fragile or unsupported claims. The report is Markdown, so it drops straight into methods sections, grants, and registered reports.

Usage

```
mp_report(x, title = "Power analysis report", manifest = TRUE)
```

Arguments

x	An mp_power (incl. mp_power_adaptive) or mp_plan() result.
title	Report title.
manifest	Include a reproducibility manifest (default TRUE).

Details

This grows mp_methods_text() and the reproducibility helpers into one artifact; pass the richer mp_plan() result to include the calibration section automatically.

Value

An object of class mp_report (a list with markdown, plus the parts: scenario, power, calibration, recommendation, warnings, manifest). Its print() method renders the Markdown.

See Also

mp_plan(), mp_methods_text(), mp_write_report().

Examples

```
if (requireNamespace("lme4", quietly = TRUE)) {
  d <- mp_design(list(subject = 12), trials_per_cell = 8)
  a <- mp_assumptions(list("(Intercept)" = 0, condition = 0.4),
                      random_effects = list(subject = list(intercept_sd = 0.5)),
                      residual_sd = 1)
  scn <- mp_scenario_lme4(y ~ condition + (1 | subject), design = d, assumptions = a)
  mp_report(mp_plan(scn, nsim = 100, seed = 1))
}
```

---

mp_report_table *Publication-ready summary table for power results*

Description

Returns a flat data frame with power estimate, CI, failure/singularity rates, and effective simulation counts. Works with mp_power, mp_sensitivity, mp_power_curve, or the result of mp_bundle_results() (uses the bundled result).

Usage

```
mp_report_table(x, ...)
```

Arguments

x	An object of class mp_power, mp_sensitivity, mp_power_curve, mp_calibration, mp_plan, or from mp_bundle_results().
...	Unused; reserved for future arguments.

Value

A data frame: for mp_power one row; for mp_calibration a one-row Type I summary; for sensitivity/curve one row per grid cell with parameter column(s), power_estimate, ci_low, ci_high, failure_rate, singular_rate, n_effective, nsim.

`mp_safeguard_effect` *Safeguard (confidence-bound) effect size from a fitted model*

Description

Computes a *safeguard* effect for power analysis: the bound of a confidence interval for a fitted effect that lies closest to zero (Perugini, Gallucci & Costantini, 2014). Planning power around this conservative, uncertainty-aware value protects against the optimism of using a noisy pilot point estimate.

Usage

```
mp_safeguard_effect(fit, term = NULL, conf_level = 0.9)
```

Arguments

`fit`	A fitted `lmer`/`glmer` model.
`term`	Fixed-effect term. Defaults to the first non-intercept effect.
`conf_level`	Two-sided confidence level for the interval (default `0.90`). Lower values are less conservative.

Details

The interval is the Wald (normal-approximation) interval from the fitted coefficient and its standard error. Pair the result with `mp_from_fit()` and `mp_sesoi()` to run a safeguard-power simulation.

Value

An object of class `mp_safeguard`: a list with `term`, `estimate`, `se`, `conf_level`, the interval (`lower`, `upper`), and the `safeguard` bound (the interval limit nearest zero, in the direction of the estimate).

See Also

`mp_sesoi()`, `mp_from_fit()`.

Examples

```
if (requireNamespace("lme4", quietly = TRUE)) {
  m <- lme4::lmer(Reaction ~ Days + (Days | Subject), data = lme4::sleepstudy)
  sg <- mp_safeguard_effect(m, term = "Days", conf_level = 0.90)
  sg
}
```

`mp_scenario` *Create a power-analysis scenario*

Description

A scenario combines: (1) a design, (2) assumptions, (3) a model specification, and (4) an analysis engine.

Usage

```
mp_scenario(
  formula,
  design,
  assumptions,
  test = c("wald", "lrt", "custom"),
  simulate_fun = NULL,
  fit_fun = NULL,
  test_fun = NULL,
  notes = NULL
)
```

Arguments

`formula`	A model formula (stored for later backends).
`design`	An `mp_design`.
`assumptions`	An `mp_assumptions`.
`test`	Character string or list identifying the test type (metadata).
`simulate_fun`	Function or NULL.
`fit_fun`	Function or NULL.
`test_fun`	Function or NULL.
`notes`	Optional free text.

Details

In Phase 1, the engine is *pluggable* via three functions:

- `simulate_fun(scenario, seed)` returns a data.frame
- `fit_fun(data, scenario)` returns a fit object
- `test_fun(fit, scenario)` returns a list with at least `p_value` (numeric scalar)

This allows `mp_power()` to run before selecting a specific backend (e.g., lme4).

Value

An object of class `mp_scenario`.

Examples

```
d <- mp_design(list(subject = 20), trials_per_cell = 5)
a <- mp_assumptions(list(condition = 0.3), residual_sd = 1)
s <- mp_scenario(y ~ condition, d, a, test = "wald")
s
```

`mp_scenario_glmmtmb_lmm`

Gaussian LMM scenario using glmmTMB

Description

Same data-generating process as `mp_scenario_lme4()` but fits with **glmmTMB** (`glmmTMB()`).

Usage

```
mp_scenario_glmmtmb_lmm(
  formula,
  design,
  assumptions,
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL,
  test_term = predictor,
  test_method = c("wald", "lrt"),
  null_formula = NULL
)
```

Arguments

`formula`	Model formula.
`design`	A `mp_design` object.
`assumptions`	A `mp_assumptions` object.
`predictor`	Predictor column name.
`subject`	Subject ID column name.
`outcome`	Outcome column name.
`item`	Optional item ID column name.
`test_term`	Term to test. A single fixed effect (default `predictor`), or a character vector of terms for an omnibus / multi-degree-of-freedom test (joint Wald for `"wald"`; for `"lrt"`/`"pb"` the `null_formula` defines the joint test).
`test_method`	Inference method: `"wald"` (default), `"satterthwaite"`, `"kenward-roger"`, `"lrt"`, or `"pb"`. See `mp_backend_lme4()`.
`null_formula`	Null-model formula required for `"lrt"` and `"pb"`.

`mp_scenario_lme4` *Create a fully specified MixPower scenario with the lme4 backend*

Description

Create a fully specified MixPower scenario with the lme4 backend

Usage

```
mp_scenario_lme4(
  formula,
  design,
  assumptions,
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL,
  test_term = predictor,
  test_method = c("wald", "lrt", "satterthwaite", "kenward-roger", "pb"),
  null_formula = NULL,
  pb_nsim = 100L,
  contrast = NULL,
  simulate = NULL
)
```

Arguments

`formula`	Model formula.
`design`	A `mp_design` object.
`assumptions`	A `mp_assumptions` object.
`predictor`	Predictor column name.
`subject`	Subject ID column name.
`outcome`	Outcome column name.
`item`	Optional item ID column name.
`test_term`	Term to test. A single fixed effect (default `predictor`), or a character vector of terms for an omnibus / multi-degree-of-freedom test (joint Wald for `"wald"`; for `"lrt"`/`"pb"` the `null_formula` defines the joint test).
`test_method`	Inference method: `"wald"` (default), `"satterthwaite"`, `"kenward-roger"`, `"lrt"`, or `"pb"`. See `mp_backend_lme4()`.
`null_formula`	Null-model formula required for `"lrt"` and `"pb"`.
`pb_nsim`	Bootstrap replicates for `test_method = "pb"` (default 100).
`contrast`	Optional named numeric vector of fixed-effect weights defining a linear contrast `L'beta` to test (e.g. weights from `emmeans`). When supplied it overrides `test_term`/`test_method` with a Wald test of the contrast.

simulate — Optional custom data-generating function `function(scenario, seed = NULL)` returning a data frame; overrides the built-in simulator (keeping the lme4 fit/test) for designs the grammar does not cover natively. Wrapped and validated by `mp_custom_dgp()`.

Value

An object of class `mp_scenario`.

`mp_scenario_lme4_binomial` *Create a fully specified MixPower scenario with the binomial lme4 backend*

Description

Create a fully specified MixPower scenario with the binomial lme4 backend

Usage

```
mp_scenario_lme4_binomial(
  formula,
  design,
  assumptions,
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL,
  test_term = predictor,
  test_method = c("wald", "lrt", "pb"),
  null_formula = NULL,
  pb_nsim = 100L
)
```

Arguments

formula	Model formula.
design	A `mp_design` object.
assumptions	A `mp_assumptions` object.
predictor	Predictor column name.
subject	Subject ID column name.
outcome	Outcome column name.
item	Optional item ID column name.
test_term	Optional explicit term to test. Defaults to `predictor`.
test_method	Inference method: `"wald"` (default), `"lrt"`, or `"pb"`.
null_formula	Null-model formula required for `"lrt"` and `"pb"`.
pb_nsim	Bootstrap replicates for `test_method = "pb"` (default 100).

Value

An object of class `mp_scenario`.

`mp_scenario_lme4_nb` *Create a fully specified MixPower scenario with the NB lme4 backend*

Description

Create a fully specified MixPower scenario with the NB lme4 backend

Usage

```
mp_scenario_lme4_nb(
  formula,
  design,
  assumptions,
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL,
  test_term = predictor,
  test_method = c("wald", "lrt", "pb"),
  null_formula = NULL,
  pb_nsim = 100L
)
```

Arguments

`formula`	Model formula.
`design`	A `mp_design` object.
`assumptions`	A `mp_assumptions` object.
`predictor`	Predictor column name.
`subject`	Subject ID column name.
`outcome`	Outcome column name.
`item`	Optional item ID column name.
`test_term`	Optional explicit term to test. Defaults to `predictor`.
`test_method`	Inference method: `"wald"` (default), `"lrt"`, or `"pb"`.
`null_formula`	Null-model formula required for `"lrt"` and `"pb"`.
`pb_nsim`	Bootstrap replicates for `test_method = "pb"` (default 100).

Value

An object of class `mp_scenario`.

`mp_scenario_lme4_poisson` *Create a fully specified MixPower scenario with the Poisson lme4 backend*

Description

Create a fully specified MixPower scenario with the Poisson lme4 backend

Usage

```
mp_scenario_lme4_poisson(
  formula,
  design,
  assumptions,
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL,
  test_term = predictor,
  test_method = c("wald", "lrt", "pb"),
  null_formula = NULL,
  pb_nsim = 100L
)
```

Arguments

`formula`	Model formula.
`design`	A `mp_design` object.
`assumptions`	A `mp_assumptions` object.
`predictor`	Predictor column name.
`subject`	Subject ID column name.
`outcome`	Outcome column name.
`item`	Optional item ID column name.
`test_term`	Optional explicit term to test. Defaults to `predictor`.
`test_method`	Inference method: `"wald"` (default), `"lrt"`, or `"pb"`.
`null_formula`	Null-model formula required for `"lrt"` and `"pb"`.
`pb_nsim`	Bootstrap replicates for `test_method = "pb"` (default 100).

Value

An object of class `mp_scenario`.

mp_sensitivity *Run power sensitivity analysis over a parameter grid*

Description

Run power sensitivity analysis over a parameter grid

Usage

```
mp_sensitivity(
  scenario,
  vary,
  nsim = 100,
  alpha = 0.05,
  seed = NULL,
  failure_policy = c("count_as_nondetect", "exclude"),
  conf_level = 0.95
)
```

Arguments

scenario	A base mp_scenario object.
vary	Named list of vectors. Names are dotted paths such as "fixed_effects.condition" or "clusters.subject".
nsim	Number of simulations for each grid cell.
alpha	Significance threshold.
seed	Optional seed for reproducible cell-wise execution.
failure_policy	Failure policy passed to mp_power().
conf_level	Confidence level passed to mp_power().

Value

An object of class mp_sensitivity.

mp_sensitivity_parallel *Parallel sensitivity analysis over a parameter grid*

Description

Like mp_sensitivity(), but evaluates each grid cell in parallel (or with a progress bar when progress = TRUE). Uses per-cell seeds seed + cell_index - 1L to match a serial ordering convention. Does not modify mp_power().

Usage

```
mp_sensitivity_parallel(
  scenario,
  vary,
  nsim = 100,
  alpha = 0.05,
  seed = NULL,
  failure_policy = c("count_as_nondetect", "exclude"),
  conf_level = 0.95,
  workers = 2L,
  progress = FALSE,
  checkpoint_dir = NULL,
  resume = TRUE,
  ...
)
```

Arguments

scenario	A base mp_scenario object.
vary	Named list of vectors. Names are dotted paths such as "fixed_effects.condition" or "clusters.subject".
nsim	Number of simulations for each grid cell.
alpha	Significance threshold.
seed	Optional seed for reproducible cell-wise execution.
failure_policy	Failure policy passed to mp_power().
conf_level	Confidence level passed to mp_power().
workers	Number of parallel workers when progress = FALSE (default 2).
progress	If TRUE, run serially with a text progress bar.
checkpoint_dir	Optional directory to save per-cell RDS results and a manifest. When resume = TRUE, existing cell files are reused if the manifest matches the current run. Use a path on **shared storage** if workers > 1.
resume	Logical; only used when checkpoint_dir is set.
...	Reserved.

Value

An object of class mp_sensitivity (same structure as mp_sensitivity()).

Note

Parallel execution requires the **parallel** package and that **mixpower** can be loaded on workers (installed package).

`mp_sesoi` *Set a smallest effect size of interest (SESOI) on a scenario*

Description

Convenience helper for planning power around a *smallest effect size of interest* rather than a single point estimate. It returns a copy of `scenario` with the focal fixed effect replaced, either by an explicit value or by scaling the current assumed effect (e.g. a conservative 15% reduction, `multiplier = 0.85`). This is the recommended way to avoid overoptimistic power based on a possibly inflated pilot/published effect (Anderson, Kelley & Maxwell, 2017; Kumle, Vo & Draschkow, 2021).

Usage

```
mp_sesoi(scenario, multiplier = 0.85, effect = NULL, term = NULL)
```

Arguments

`scenario`	An `mp_scenario` object.
`multiplier`	Numeric factor applied to the current fixed effect when `effect` is not supplied (default `0.85`, a 15% reduction).
`effect`	Optional explicit SESOI on the model's coefficient scale. When supplied it overrides `multiplier`. May be a numeric scalar or an `mp_safeguard_effect()` result.
`term`	Fixed-effect term to modify. Defaults to the scenario's test term (or the first non-intercept fixed effect).

Value

A modified `mp_scenario` object.

See Also

`mp_safeguard_effect()` for a data-driven conservative effect.

Examples

```
d <- mp_design(list(subject = 30), trials_per_cell = 8)
a <- mp_assumptions(
  fixed_effects = list("(Intercept)" = 0, condition = 0.5),
  random_effects = list(subject = list(intercept_sd = 0.5)),
  residual_sd = 1
)
scn <- mp_scenario_lme4(y ~ condition + (1 | subject), design = d, assumptions = a)
# Power for an effect 15% smaller than assumed:
scn_sesoi <- mp_sesoi(scn, multiplier = 0.85)
scn_sesoi$assumptions$fixed_effects$condition
```

`mp_solve_sample_size` *Solve for minimum sample size achieving target power*

Description

Evaluates power on a user-supplied grid of values for one parameter (e.g. cluster size) via `mp_power_curve()`, then returns the smallest grid value whose power estimate meets or exceeds the target. Diagnostics (failure rate, singular rate, n_effective) are exposed in the returned `results` table.

Usage

```
mp_solve_sample_size(
  scenario,
  parameter,
  grid,
  target_power = 0.8,
  nsim = 100,
  alpha = 0.05,
  seed = NULL,
  failure_policy = c("count_as_nondetect", "exclude"),
  conf_level = 0.95
)
```

Arguments

`scenario`	An `mp_scenario`.
`parameter`	Dotted path of the single parameter to vary (e.g. `"clusters.subject"`).
`grid`	Numeric vector of candidate values. Use `mp_grid_sample_size()` to build a grid from bounds and either `length.out` or `by`.
`target_power`	Target power threshold (default 0.8).
`nsim`	Number of simulations per grid point (default 100).
`alpha`	Significance level (default 0.05).
`seed`	Optional seed for reproducibility.
`failure_policy`	How to treat failed fits: `"count_as_nondetect"` or `"exclude"`.
`conf_level`	Confidence level for power intervals (default 0.95).

Value

A list with `target_power`, `parameter`, `solution` (numeric: minimum grid value achieving target power, or `NA` if none), and `results` (data frame with estimate, failure_rate, singular_rate, n_effective, etc., per grid point).

mp_stop *Stopping rule for adaptive power simulation*

Description

Specifies when mp_power_adaptive() should stop adding simulation batches. Stopping is triggered (after at least min_nsim replicates) when *either*:

Usage

```
mp_stop(
  ci_halfwidth = 0.02,
  target = NULL,
  min_nsim = 200,
  max_nsim = 5000,
  batch = 200
)
```

Arguments

ci_halfwidth	Target half-width of the power CI (e.g. 0.02), or NULL to use only the decision rule.
target	Optional target power (in (0, 1)); enables the decision rule.
min_nsim	Minimum replicates before stopping is considered (default 200).
max_nsim	Maximum replicates (hard cap, default 5000).
batch	Replicates added per iteration (default 200).

Details

- **precision**: the power confidence interval's half-width is <= ci_halfwidth, or
- **decision**: a target power is given and the interval lies entirely above or entirely below it (the accept/reject decision is unambiguous).

Otherwise it keeps adding batches until max_nsim is reached.

Value

An object of class mp_stop.

See Also

mp_power_adaptive().

Examples

```
mp_stop(ci_halfwidth = 0.02, max_nsim = 4000)
mp_stop(target = 0.8, ci_halfwidth = 0.03)
```

mp_write_report *Write a power-analysis report to a file*

Description

Writes an `mp_report()` to disk as Markdown (default) or HTML. HTML rendering uses `rmarkdown`/pandoc when available; otherwise use `"markdown"` (which is easily converted to HTML/PDF by any Markdown tool).

Usage

```
mp_write_report(report, file, format = c("markdown", "html"))
```

Arguments

`report`	An `mp_report` object.
`file`	Output file path.
`format`	`"markdown"` (default) or `"html"`.

Value

Invisibly, `file`.

See Also

`mp_report()`.

mp_write_results *Write results or bundle to CSV or JSON*

Description

Writes the report table (and for bundles, manifest/labels) to file. CSV writes the publication-ready table only; JSON writes report table plus manifest and labels when `x` is an `mp_bundle`.

Usage

```
mp_write_results(x, file, format = c("csv", "json"), ...)
```

Arguments

`x`	An object from `mp_bundle_results()`, or `mp_power`, `mp_sensitivity`, or `mp_power_curve`.
`file`	Path to output file (extension need not match format).
`format`	`"csv"` or `"json"`.
`...`	For CSV, arguments passed to `utils::write.csv()` (e.g. `row.names = FALSE`).

Value

Invisibly the path `file`.

---

plot.mp_power *Plot the p-value distribution of a power analysis*

Description

A histogram of the per-replicate p-values from `mp_power()`, with the `alpha` threshold marked. The shaded area to the left of `alpha` is the estimated power. Requires a run with `aggregate = "full"` (the default), which retains per-replicate p-values.

Usage

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

Arguments

`x`	An `mp_power` object.
`...`	Passed to `graphics::hist()`.

Value

Invisibly, the p-values plotted.

---

plot.mp_power_curve *Plot a power curve*

Description

Plot a power curve

Usage

```
## S3 method for class 'mp_power_curve'
plot(x, y = c("estimate", "failure_rate", "singular_rate", "n_effective"), ...)
```

Arguments

`x`	An `mp_power_curve` object.
`y`	What to plot on the y-axis: `"estimate"` (power), `"failure_rate"`, `"singular_rate"`, or `"n_effective"`.
`...`	Arguments passed to `graphics::plot()`.

Value

Invisibly returns the plotted data.

plot.mp_sensitivity *Plot a sensitivity analysis*

Description

For one varying parameter: line plot with optional CI segments when `y = "estimate"`. For two varying parameters: heatmap. More than two parameters is not supported.

Usage

```
## S3 method for class 'mp_sensitivity'
plot(x, y = c("estimate", "failure_rate", "singular_rate", "n_effective"), ...)
```

Arguments

`x`	An `mp_sensitivity` object.
`y`	What to plot: `"estimate"` (power), `"failure_rate"`, `"singular_rate"`, or `"n_effective"`.
`...`	Additional graphical arguments passed to `graphics::plot()` (1D) or `graphics::image()` (2D).

Value

Invisibly returns the plotted data (1D: ordered data frame; 2D: matrix).

plot_power *Plot power results*

Description

Plot power results

Usage

```
plot_power(results, ...)
```

Arguments

`results`	A data.frame with effect and power columns.
`...`	Additional arguments passed to plot.

Value

Invisibly returns the plot data.

run_parallel *Placeholder for parallel execution*

Description

Placeholder for parallel execution

Usage

```
run_parallel(fun, ...)
```

Arguments

fun	Function to run.
...	Additional arguments to pass to fun.

Value

The result of fun.

simulate_glmm_binomial_data *Simulate binary outcome data for a GLMM with random effects*

Description

Thin wrapper over the shared simulation engine (.mp_simulate_mixed) with a logit link and Bernoulli response. Random-effect sizes (intercept and optional slope on predictor) come from scenario$assumptions$random_effects.

Usage

```
simulate_glmm_binomial_data(
  scenario,
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL
)
```

Arguments

scenario	An mp_scenario object.
predictor	Predictor column name.
subject	Subject ID column name.
outcome	Outcome column name.
item	Optional item ID column name.

Value

A data.frame with outcome and predictors.

`simulate_glmm_nb_data` *Simulate count outcome data for a Negative Binomial GLMM with random effects*

Description

Thin wrapper over the shared simulation engine (`.mp_simulate_mixed`) with a log link and negative-binomial response. Random-effect sizes (intercept and optional slope on `predictor`) come from `scenario$assumptions$random_effects`.

Usage

```
simulate_glmm_nb_data(
  scenario,
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL,
  theta = NULL
)
```

Arguments

`scenario`	An `mp_scenario` object.
`predictor`	Predictor column name.
`subject`	Subject ID column name.
`outcome`	Outcome column name.
`item`	Optional item ID column name.
`theta`	NB dispersion parameter (size); larger means less over-dispersion. Defaults to `scenario$assumptions$theta` or 1.

Value

A data.frame with outcome and predictors.

simulate_glmm_poisson_data *Simulate count outcome data for a Poisson GLMM with random effects*

Description

Thin wrapper over the shared simulation engine (`.mp_simulate_mixed`) with a log link and Poisson response. Random-effect sizes (intercept and optional slope on `predictor`) come from `scenario$assumptions$random_ef`

Usage

```
simulate_glmm_poisson_data(
  scenario,
  predictor = "condition",
  subject = "subject",
  outcome = "y",
  item = NULL
)
```

Arguments

`scenario`	An `mp_scenario` object.
`predictor`	Predictor column name.
`subject`	Subject ID column name.
`outcome`	Outcome column name.
`item`	Optional item ID column name.

Value

A data.frame with outcome and predictors.

simulate_power *Run a simple simulation-based power study*

Description

Run a simple simulation-based power study

Usage

```
simulate_power(scenario, nsim = 100, seed = NULL)
```

Arguments

scenario	A scenario object.
nsim	Number of simulations.
seed	Optional random seed.

Value

A data.frame of simulated p-values.

---

summarize_simulations *Summarize simulation outputs*

Description

Summarize simulation outputs

Usage

```
summarize_simulations(simulations)
```

Arguments

simulations	A data.frame of simulations.

Value

A summary data.frame.

---

test_effect *Extract a test statistic for a model term*

Description

Extract a test statistic for a model term

Usage

```
test_effect(fit, term)
```

Arguments

fit	A fitted model object.
term	Term name to test.

Value

A data.frame with coefficient information.

validate_mp_backend *Validate a MixPower backend*

Description

Checks that simulate_fun, fit_fun, and test_fun are functions and that their formal arguments are compatible with what mp_power() and .run_one_rep() invoke. Does not run simulations.

Usage

```
validate_mp_backend(engine)
```

Arguments

engine	A mp_backend object or a plain list with simulate_fun, fit_fun, and test_fun.

Value

Invisibly TRUE if valid; otherwise throws an error.

Index

as.data.frame(), *26*
as_tibble.mp_power, 5
autoplot.mp_sensitivity, 6

effect_size, 6

fit_model, 8

graphics::hist(), *54*
graphics::image(), *55*
graphics::plot(), *54, 55*

mixpower *(*mixpower-package*)*, 3
mixpower-package, 3
mp_assumptions, 8
mp_assumptions(), *6*, *22, 23*, *37*
mp_attach_calibration, 10
mp_backend, 10
mp_backend_glmmtmb, 11
mp_backend_lme4, 12
mp_backend_lme4(), *11*, *43, 44*
mp_backend_lme4_binomial, 13
mp_backend_lme4_nb, 14
mp_backend_lme4_poisson, 15
mp_beta_to_d *(*effect_size*)*, 6
mp_beta_to_r2 *(*effect_size*)*, 6
mp_bundle_results, 15
mp_bundle_results(), *40*, *53*
mp_calibrate, 16
mp_calibrate(), *10*, *20*, *29–31*, *38, 39*
mp_compare_models, 18
mp_compare_models(), *20*
mp_custom_dgp, 19
mp_custom_dgp(), *45*
mp_d_to_beta *(*effect_size*)*, 6
mp_d_to_beta(), *23*
mp_design, 20
mp_design(), *5*, *23*
mp_elicit, 22
mp_extend, 23
mp_extend(), *25*
mp_f_to_beta *(*effect_size*)*, 6
mp_from_fit, 24
mp_from_fit(), *23, 24*, *41*
mp_grid_sample_size, 25
mp_grid_sample_size(), *51*
mp_icc_to_sd *(*effect_size*)*, 6
mp_icc_to_sd(), *23*
mp_logodds_to_or *(*effect_size*)*, 6
mp_manifest, 26
mp_manifest(), *16*
mp_methods_text, 27
mp_methods_text(), *39, 40*
mp_missing, 27
mp_missing(), *23*
mp_or_to_logodds *(*effect_size*)*, 6
mp_plan, 29
mp_plan(), *10*, *31*, *39, 40*
mp_power, *15*, 30, *40*
mp_power(), *5*, *10*, *17, 18*, *24*, *27*, *29, 30*, *32–38*, *48, 49*, *54*, *60*
mp_power_adaptive, 32
mp_power_adaptive(), *52*
mp_power_checkpoint, 33
mp_power_checkpoint(), *33*
mp_power_curve, *15*, 34, *40*
mp_power_curve(), *24*, *35, 36*, *51*
mp_power_curve_parallel, 35
mp_quick_power, 36
mp_quick_power(), *5*
mp_r2_to_beta *(*effect_size*)*, 6
mp_recommend_method, 38
mp_recommend_method(), *17*, *29, 30*
mp_report, 39
mp_report(), *53*
mp_report_table, 40
mp_safeguard_effect, 41
mp_safeguard_effect(), *50*
mp_scenario, 42

mp_scenario(), *20*
mp_scenario_glmmtmb_lmm, 43
mp_scenario_lme4, 44
mp_scenario_lme4(), *20*, *43*
mp_scenario_lme4_binomial, 45
mp_scenario_lme4_nb, 46
mp_scenario_lme4_poisson, 47
mp_sd_to_icc *(*effect_size*)*, 6
mp_sensitivity, *15*, *40*, 48
mp_sensitivity(), *34*, *48, 49*
mp_sensitivity_parallel, 48
mp_sesoi, 50
mp_sesoi(), *41*
mp_solve_sample_size, 51
mp_solve_sample_size(), *24, 25*
mp_stop, 52
mp_stop(), *32, 33*
mp_t_to_beta *(*effect_size*)*, 6
mp_write_report, 53
mp_write_report(), *40*
mp_write_results, 53

plot.mp_power, 54
plot.mp_power_curve, 54
plot.mp_power_curve(), *6*
plot.mp_sensitivity, 55
plot.mp_sensitivity(), *6*
plot_power, 55

run_parallel, 56

seq(), *26*
simulate_glmm_binomial_data, 56
simulate_glmm_nb_data, 57
simulate_glmm_poisson_data, 58
simulate_power, 58
stats::simulate(), *24*
summarize_simulations, 59

test_effect, 59
tibble::as_tibble(), *5*
tibble::tibble(), *5*

utils::write.csv(), *53*

validate_mp_backend, 60