Package: PowRPriori (via r-universe)

May 12, 2026

Title Power Analysis via Data Simulation for (Generalized) Linear Mixed Effects Models

Version 0.2.0

Description Conduct a priori power analyses via Monte-Carlo style data simulation for linear and generalized linear mixed-effects models (LMMs/GLMMs). Provides a user-friendly workflow with helper functions to easily define fixed and random effects as well as diagnostic functions to evaluate the adequacy of the results of the power analysis.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports dplyr, doFuture, foreach, future, ggplot2, lme4, lmerTest, magrittr, MASS, purrr, rlang, scales, stats, tidyr, tidyselect, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

URL https://github.com/mirgll/PowRPriori

BugReports https://github.com/mirgll/PowRPriori/issues

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake make libicu-dev

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

Date/Publication 2026-05-12 11:37:32 UTC

RemoteUrl https://github.com/mirgll/powrpriori

RemoteRef HEAD

RemoteSha 8465baae15892c652f9d5d4a1b1a1f0c1e9ab2ec

Contents

`.center_predictors` *Internal Helper Function for Mean-Centering*

Description

This helper function is responsible for the mean centering of predictors. It detects whether any given predictor is repeated within clusters (e.g. repeated measurements in participants) and automatically applies within-cluster mean centering if this is the case. In all other cases, grand mean centering is applied.

Usage

```
.center_predictors(df, formula, subject_id_col = ".PowR_id")
```

Arguments

`df`	The dataframe to process
`formula`	The lme4 formula to extract predictors
`subject_id_col`	The name of the subject identifier column

Value

Dataframe with centered predictors

.create_design_matrix *Create the Design Matrix for a Simulation*

Description

An internal helper function that takes the design specification and an lme4-style formula to generate a design matrix representing the structure of the experimental design.

Usage

```
.create_design_matrix(design, formula = NULL)
```

Arguments

design	A PowRPriori_design object from define_design().
formula	The lme4-style model formula.

Value

A tibble (data frame) with predictor variables.

.plot_data *Internal Data Plotting Engine*

Description

An internal helper function containing the logic to "intelligently" create plots from simulated data. It automatically chooses between spaghetti plots and jitter/point-range plots depending on the specified design and model family.It derives sensible defaults for plot aesthetics from the design, if they are not supplied directly via the plot_sim_model function.

Usage

```
.plot_data(
  data,
  design,
  formula,
  family,
  x_var,
  group_var,
  color_var,
  facet_var,
  n_data_points
)
```

Arguments

data	The data frame to plot.
design	The PowRPriori_design object.
formula	An lme4-style formula (e.g. outcome ~ predictor1 * predictor2 + (1 | subject))
family	The model family. Defaults to "gaussian", other possible values are "binomial" or "poisson".
x_var, group_var, color_var, facet_var	Strings specifying variables for plot aesthetics.
n_data_points	The maximum number of trajectories in spaghetti plots.

Value

A ggplot object.

.simulate_outcome *Simulate the Outcome Variable*

Description

An internal helper function that takes a complete design matrix and simulates the dependent variable based on the specified fixed and random effects.

Usage

```
.simulate_outcome(
  design_df,
  formula,
  fixed_effects,
  sds_random,
  family = "gaussian"
)
```

Arguments

design_df	The data frame from .create_design_matrix.
formula	The model formula.
fixed_effects	A list of the fixed effects coefficients.
sds_random	A list of the random effects' standard deviations and correlations.
family	A string indicating the model family.

Value

The input design_df with an added column for the outcome variable.

.to_factor_safely *Safely Convert Character Vectors to Factors*

Description

An internal helper function that converts a character vector to a factor, ensuring the level order is based on the first appearance of each element. If the input is not a character vector, it's returned unchanged.

Usage

```
.to_factor_safely(x)
```

Arguments

x	A vector from a design specification.

Value

A factor with levels in order of appearance, or the original object.

define_design *Define the Structure of a Planned Experimental Design*

Description

This is the primary setup function for any power simulation in PowRPriori. It creates a special PowRPriori_design object that contains all the necessary information about the variables, the hierarchical structure, and the sample size of a planned study.

Usage

```
define_design(sample_size, between = NULL, within = NULL)
```

Arguments

sample_size	A named list specifying the building blocks and dimensions of the planned study sample (e.g. list(class = 10, pupil = 20)). This dictates which analysis units exist and the number of elements within each unit.
between	A list of between-subject variables. These are variables where a given unit is assigned to exactly one level of the variable (e.g., participants or entire clusters being assigned to either an intervention or a control group).
within	A list of within-subject variables. These are variables where all levels are observed within the same unit (e.g., repeated pre- and post-measurements within participants).

Details

Variable Specification: Variables can be specified in different formats depending on their scale: Nominal variables (e.g. a group variable with levels "control" and "treatment") can be specified as factors (`group = factor(c("control", "treatment"))`) or as character vectors (`group = c("control", "treatment")`), which are automatically converted to factors. Continuous variables can be specified via their expected mean and standard deviation (`test_score = list(mean = 10, sd = 5)`). Additionally, variables can be defined as fixed numerical vectors (`predictor = 1:4`).

Assignment of Variables: By default, if `between` variables are specified directly as a simple list (e.g., `between = list(treatment = c("A", "B"))`), they are randomized at the lowest level of the design (individual assignment). If a `between` variable should be assigned at a higher cluster level (e.g., cluster-randomization at the class level), it must be wrapped in a named list corresponding to that specific analysis unit. You do not need to mimic the full hierarchical structure of your design here (e.g., no need to write school = list(class = list(...))). Simply wrap the predictor in a single list named after the exact cluster level it belongs to (see the nested design example below).

`within` variables, on the other hand, are always crossed with the level-1 analysis units, effectively creating repeated measures for the lowest level.

For a full tutorial and more complex design structures, see the package vignette: `vignette("Workflow-Example", package = "PowRPriori")`.

Value

A `PowRPriori_design` object containing the parsed design specifications.

Examples

```
# Simple 2x2 mixed design
simple_design <- define_design(
  sample_size = list(subject = 20),
  between = list(group = c("Control", "Treatment")),
  within = list(time = c("pre", "post"))
)

# A nested (cluster-randomized) design where the intervention
# is assigned at the class level.
nested_design <- define_design(
  sample_size = list(class = 10,
                     pupil = 20),
  between = list(
    class = list(intervention = c("yes", "no")),
    pupil = list(support = c("yes", "no"))
  )
)
```

`fixed_effects_from_average_outcome`

Calculate Fixed-Effects Coefficients from Mean Outcomes

Description

A user-friendly helper function to translate expected outcomes (e.g., cell means, probabilities, or rates) into the regression coefficients required by the simulation. This is often more intuitive than specifying coefficients directly.

Usage

```
fixed_effects_from_average_outcome(
  formula,
  outcome,
  center = TRUE,
  family = "gaussian"
)
```

Arguments

formula	The fixed-effects part of the model formula (e.g., y ~ group * time).
outcome	A data frame containing columns for all predictor variables and exactly one column for the expected outcome values.
center	If TRUE (default), predictors in the design grid are effect-coded (i.e. the contrast coding is changed so that they are treated as being centered) to yield a grand-mean intercept. Additionally, the output is flagged to instruct power_sim() to automatically center the simulated data. If set to FALSE, the predictors are treated as not centered and no alterations to the contrast coding are applied.
family	The model family ("gaussian", "binomial", "poisson"). The outcome values should be means for gaussian, probabilities (0-1) for binomial, and non-negative rates/counts for poisson.

Details

By default, this function applies effect coding (orthogonal contrasts) to the predictors in your design (center = TRUE). Because the predictors are treated as being mean-centered this way, the intercept of the calculated coefficients represents the grand mean of your specified outcomes.

Importantly, setting center = TRUE also adds an attribute to the resulting coefficient list. When the coefficients are later passed to power_sim(), the simulation engine will detect this attribute and automatically apply the corresponding mean-centering to the generated sample data (grand-mean centering for between-subject variables, and within-cluster / person-mean centering for within-subject variables).

The choices for these respective centering techniques have been made to ensure the most robust fixed / random effects and power estimates. For more details on the issue of centering, refer to the vignette: vignette("Workflow-Example", package = "PowRPriori")

Value

A named list of coefficients suitable for the fixed_effects argument in power_sim().

Examples

```
outcome_means <- tidyr::expand_grid(
  group = c("Control", "Treatment"),
  time = c("pre", "post")
)
outcome_means$mean <- c(10, 10, 12, 15) # Specify expected means

#Per default, the predictors are effect-coded (centered) here
fixed_effects_from_average_outcome(
  formula = score ~ group * time,
  outcome = outcome_means
)
```

---

get_fixed_effects_structure

Get the Expected Fixed-Effects Structure

Description

Analyzes a model formula and a design object to generate a template for the `fixed_effects` parameter. This is a helper function designed to prevent typos and ensure all necessary coefficients are specified. By default, this function prints a copy-paste-able code snippet to the console, where the user only needs to fill in placeholders (`...`) for the values.

Usage

```
get_fixed_effects_structure(formula, design)
```

Arguments

`formula`	An lme4-style model formula (e.g. `outcome ~ predictor1 * predictor2 + (1 | id)`). Since this function only uses the fixed-effects part of the model, specifying the random effects is optional here.
`design`	A `PowRPriori_design` object created with `define_design()`.

Value

Invisibly returns a named list with placeholders, which can be used as a template for the `fixed_effects` argument in `power_sim()`.

Examples

```
design <- define_design(
  sample_size = list(subject = 20),
  between = list(group = c("Control", "Treatment")),
  within = list(time = c("pre", "post"))
)
get_fixed_effects_structure(y ~ group * time, design)
```

get_random_effects_structure *Get the Expected Random-Effects Structure*

Description

Analyzes the random effects terms in a model formula and generates a template for the specified `random_effects` parameters. This helps in specifying the required standard deviations and correlations correctly. By default, this function prints a copy-paste-able code snippet to the console, where the user only needs to fill in placeholders (`...`) for the values.

Usage

```
get_random_effects_structure(formula, design, family = "gaussian")
```

Arguments

`formula`	An lme4-style model formula (e.g. `outcome ~ predictor1 * predictor2 + (1 | id)`).
`design`	A `PowRPriori_design` object created with `define_design()`.
`family`	The model family (`"gaussian"`, `"binomial"`, `"poisson"`). Determines how the residual variance in a design needs to be handled. Defaults to `"gaussian"`.

Value

Invisibly returns a nested list with placeholders, serving as a template for the `random_effects` argument in `power_sim()`.

Examples

```
design <- define_design(
  sample_size = list(subject = 20),
  within = list(time = c("pre", "post"))
)
get_random_effects_structure(y ~ time + (time|subject), design, family = "gaussian")
```

plot_sim_model *Visualize Simulation Data or Power Simulation Results*

Description

Generic plotting function with methods for different objects.

- When used on an lme4-style `formula`, it simulates and plots a single plausible dataset.
- When used on a `PowRPriori` object, it plots either a power curve from the object or a dataset from the simulation.

The plotting of the dataset is designed to aid in evaluating whether the simulated data is plausible in the context of the desired study design and model specifications. It can help determine whether the chosen parameters are sensible or might need some adapting. The power curve, plotted from the resulting `PowRPriori` object of the `power_sim` function visualizes the iterations of the simulation across the different sample sizes for which the power was calculated during simulation.

Usage

```
plot_sim_model(
  object,
  type,
  design,
  fixed_effects,
  random_effects,
  family,
  center,
  n,
  x_var,
  group_var,
  color_var,
  facet_var,
  n_data_points,
  ...
)

## S3 method for class 'formula'
plot_sim_model(
  object,
  type = "data",
  design,
  fixed_effects,
  random_effects,
  family = "gaussian",
  center = "auto",
  n = NULL,
  x_var = NULL,
  group_var = NULL,
  color_var = NULL,
  facet_var = NULL,
  n_data_points = 10,
  ...
)
```

```
## S3 method for class 'PowRPriori'
plot_sim_model(
  object,
  type = "power_curve",
  design = NULL,
  fixed_effects = NULL,
  random_effects = NULL,
  family = NULL,
  center = NULL,
  n = NULL,
  x_var = NULL,
  group_var = NULL,
  color_var = NULL,
  facet_var = NULL,
  n_data_points = 10,
  ...
)
```

Arguments

`object`	The object to base the plot on. Can be either a `PowRPriori` object or an `lme4`-style formula
`type`	The type of plot to create: `"power_curve"` (default) or `"data"` (to visualize the sample data from the simulation).
`design`	A `PowRPriori_design` object.
`fixed_effects, random_effects`	Lists of effect parameters.
`family`	The model family. Defaults to `"gaussian"`, other possible values are `"binomial"` or `"poisson"`.
`center`	Controls if centering is applied to the predictors prior to plotting. Defaults to `"auto"`.
`n`	The total sample size to simulate for the plot (overwrites the lowest design level).
`x_var, group_var, color_var, facet_var`	Strings specifying variables for plot aesthetics.
`n_data_points`	The maximum number of trajectories in spaghetti plots.
`...`	Additional arguments (not used).

Details

The parameters `x_var, group_var, color_var and facet_var` are `NULL` by default. If left `NULL`, they are automatically extracted from the `PowRPriori` object or the `design` object.

Value

A `ggplot` object.

Examples

```
# 1. Plot prior to simulation to check data plausibility
design <- define_design(
  sample_size = list(subject = 30),
  between = list(group = c("Control", "Treatment")),
  within = list(time = c("pre", "post"))
)

fixed_effects <- list(
  `(Intercept)` = 10,
  groupTreatment = 2,
  timepost = 1,
  `groupTreatment:timepost` = 3
)

random_effects <- list(
  subject = list(`(Intercept)` = 3),
  sd_resid = 3
)

plot_sim_model(
  y ~ group * time + (1|subject),
  design = design,
  fixed_effects = fixed_effects,
  random_effects = random_effects
)

# 2. Plot from PowRPriori object after simulation
  power_results <- power_sim(
    formula = y ~ group * time + (1|subject),
    design = design,
    fixed_effects = fixed_effects,
    random_effects = random_effects,
    test_parameter = "groupTreatment:timepost",
    center = TRUE,
    n_start = 100,
    n_increment = 5,
    n_sims = 10, # Using a smaller n_sims for a quick example
    max_simulation_steps = 1, # Using this for a quick example as well
    parallel_plan = "sequential"
  )

  # Power curve
  plot_sim_model(power_results, type = "power_curve")

  # Plot sample data with automated aesthetics extraction
  plot_sim_model(power_results, type = "data")
```

power_sim *Perform a Power Analysis for (Generalized) Linear Mixed-Effects Models via Data Simulation*

Description

This is the main function of the PowRPriori package. It iteratively simulates datasets for increasing sample sizes to determine the required sample size to achieve a desired level of statistical power for specific model parameters.

Usage

```
power_sim(
  formula,
  design,
  test_parameter = NULL,
  fixed_effects,
  random_effects = NULL,
  icc_specs = NULL,
  overall_variance = NULL,
  family = "gaussian",
  adjust_p_value = "BH",
  center = "auto",
  power_crit = 0.8,
  along = NULL,
  n_start = NULL,
  n_increment,
  max_simulation_steps = 100,
  n_issue_stop_prop = 0.2,
  n_sims = 2000,
  alpha = 0.05,
  parallel_plan = "multisession"
)
```

Arguments

formula	An lme4-style model formula (e.g. outcome ~ predictor1 * predictor2 + (1 | id)).
design	A PowRPriori_design object created by define_design().
test_parameter	A character vector of the variable names to test for power. If NULL (default), power is calculated for all fixed effects except the intercept. Note: The parameter names need to comply with the names expected by the model. Correctly naming the variables is aided by the output of the get_fixed_effects_structure() helper function.
fixed_effects	A named list of the fixed-effects coefficients. It is highly recommended to generate this using get_fixed_effects_structure() or fixed_effects_from_average_outcome().

`random_effects` A named, nested list specifying the standard deviations (SDs) and (if applicable) correlations of the random effects. It is highly recommended to generate this using `get_random_effects_structure()`. If this parameter is not used, `icc_specs` and `overall_variance` need to be supplied.

`icc_specs` Optional. A named list of Intraclass Correlation Coefficients for defining simple random-intercept models. Must be used with `overall_variance`.

`overall_variance` The total variance of the outcome, required when `icc_specs` is used.

`family` The model family: `"gaussian"` (for LMMs), `"binomial"` (for logistic GLMMs), or `"poisson"` (for poisson GLMMs).

`adjust_p_value` Controls how p-values in the data simulation are adjusted when power is calculated for more than one parameter (as specified in `test_parameter`). Possible values are the same as in the function `p.adjust`. Defaults to "BH", which is the Benjamini-Hochberg correction. Setting this to `FALSE` disables p-value adjustment, although this is discouraged.

`center` Controls if the simulation should automatically center predictors. Defaults to `"auto"`, which extracts the centering attribute (if present) from the `fixed_effects` list. Set to `TRUE` to force mean-centering, or `FALSE` to disable it.

`power_crit` The desired statistical power level (e.g., 0.80 for 80%).

`along` A string specifying the sample size variable that the power analysis should be based on. Must be present in the `sample_size` variable of the `design` parameter.

`n_start` The starting sample size for the simulation. Defaults to `NULL`, in which case the number of the `along` parameter specified in the `design` parameter object is used.

`n_increment` The step size for increasing the sample size in each iteration.

`max_simulation_steps` A hard stop for the simulation, limiting the number of sample size steps to prevent infinite loops. Defaults to 100 steps.

`n_issue_stop_prop` The proportion of model issues (e.g., singular fits, non-convergence) at which the simulation will be automatically canceled. Defaults to a proportion of 20%.

`n_sims` The number of simulations to run for each sample size step. Defaults to 2000.

`alpha` The significance level (alpha) for the power calculation. Defaults to 0.05.

`parallel_plan` A string specifying the `future` plan for parallel processing. Defaults to `"multisession"` to enable parallel computing. Use `"sequential"` for debugging.

Details

The function supports parallel computation using `future`. Simple linear models (i.e. regression models) can also be analyzed using this function. In this case, no specification of the `random_effects` or `icc_specs` parameter is necessary.`icc_specs` should only be used when simulating a model containing only random intercepts and no random slopes. Refer to the vignette for a more detailed description of the complete workflow for using this function.

Value

An object of class PowRPriori, which is a list containing the power table, a sample dataset, all simulation parameters, and detailed results from all runs (coefficients and random effect estimates).

Examples

```
design <- define_design(
  sample_size = list(subject = 20),
  between = list(group = c("Control", "Treatment")),
  within = list(time = c("pre", "post"))
)

fixed_effects <- list(
  `(Intercept)` = 10,
  groupTreatment = 2,
  timepost = 1,
  `groupTreatment:timepost` = 1.5
)

random_effects <- list(
  subject = list(`(Intercept)` = 3),
  sd_resid = 5
)

power_results <- power_sim(
  formula = y ~ group * time + (1|subject),
  design = design,
  fixed_effects = fixed_effects,
  random_effects = random_effects,
  test_parameter = "groupTreatment:timepost",
  center = TRUE,
  n_increment = 5,
  n_start = 100,
  n_sims = 10, # Use low n_sims for quick examples
  max_simulation_steps = 1, # Used here for a quick example as well
  parallel_plan = "sequential"
)

summary(power_results)
plot_sim_model(power_results)
```

---

summary.PowRPriori *Summarize a Power Simulation Result*

Description

Provides a detailed and context-aware summary of a PowRPriori object. The output includes the power table, parameter recovery diagnostics for fixed and random effects, and (if applicable) calculated Intra-Class Correlations (ICCs). The output is tailored for different model types (LM, LMM, GLMM).

Usage

```
## S3 method for class 'PowRPriori'
summary(object, ...)
```

Arguments

`object`	An object of class `PowRPriori` returned by `power_sim()`.
`...`	Additional arguments (not used).

Value

Prints a formatted summary to the console.

Index