

Package ‘PowRPriori’

December 14, 2025

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

Version 0.1.1

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

RoxygenNote 7.3.2

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

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

NeedsCompilation no

Author Markus Grill [aut, cre]

Maintainer Markus Grill <markus.grill@uni-wh.de>

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2025-12-14 16:50:19 UTC

Contents

.create_design_matrix	2
.plot_data	3

<code>.simulate_outcome</code>	4
<code>.to_factor_safely</code>	4
<code>define_design</code>	5
<code>fixed_effects_from_average_outcome</code>	6
<code>get_fixed_effects_structure</code>	7
<code>get_random_effects_structure</code>	8
<code>plot_sim_model</code>	8
<code>power_sim</code>	11
<code>summary.PowRPriori</code>	14

.create_design_matrix *Create the Design Matrix for a Simulation***Description**

An internal helper function that takes the design specification and a sample size to generate a single data frame representing all observations for one simulation run. It handles within-, between-, nested, and crossed factors.

Usage

```
.create_design_matrix(design, current_n, n_is_total = TRUE)
```

Arguments

<code>design</code>	A PowRPriori_design object from <code>define_design()</code> .
<code>current_n</code>	The sample size for which the design matrix should be generated.
<code>n_is_total</code>	A boolean that controls how <code>current_n</code> is interpreted. TRUE assumes that the whole sample used for the simulation should be size <code>current_n</code> , FALSE assumes that <code>current_n</code> specifies the size of each cell as defined by <code>design</code> .

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

<code>data</code>	The data frame to plot.
<code>design</code>	The PowRPriori_design object.
<code>formula</code>	An lme4-style formula (e.g. <code>outcome ~ predictor1 * predictor2 + (1 subject)</code>)
<code>family</code>	The model family (e.g., "gaussian").
<code>x_var, group_var, color_var, facet_var</code>	Strings specifying variables for plot aesthetics.
<code>n_data_points</code>	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

<code>design_df</code>	The data frame from <code>.create_design_matrix</code> .
<code>formula</code>	The model formula.
<code>fixed_effects</code>	A list of the fixed effects coefficients.
<code>sds_random</code>	A list of the random effects' standard deviations and correlations.
<code>family</code>	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 Experimental Structure of an Experimental Design

Description

This is the primary setup function for any power simulation. It creates a special PowRPriori_design object that contains all the necessary information about the variables and structure of your study.

Usage

```
define_design(id, between = NULL, within = NULL, nesting_vars = NULL)
```

Arguments

- | | |
|--------------|--|
| id | A string specifying the name of the lowest-level unit of observation (e.g., "subject", "pupil", plot_of_land). |
| between | A list of between-subject factors. Can be a simple list (for individual assignment) or a nested list (e.g., list(class = list(group = ...))) for group-level assignment. |
| within | A list of within-subject factors. Each id will be measured at every level of these factors. |
| nesting_vars | A list of variables that are only used for grouping in the random effects structure (e.g., (1 school/class)). |

Details

Variables can be specified as different types. Nominal scale variables (e.g. group with levels "control" and "treatment") can be specified as factors (group = factor(c("control", "treatment"))) or as character vectors (c("control", "treatment")), in which case they are automatically converted to factors later on. Continuous variables can be specified via mean and standard deviation (test_score = list(mean = 10, sd = 5)). Additionally, variables can also be defined as numerical vectors (predictor = 1:4).

The between argument is particularly flexible. For simple designs, you can provide a "flat" list of factors. For complex designs like cluster-randomized trials, you can provide a hierarchical list to specify the level of assignment (see examples). For a full tutorial, see the package vignette: vignette("Workflow-Example", package = "PowRPriori")

Value

A PowRPriori_design object, which is a list containing the design specifications.

Examples

```
# Simple 2x2 mixed design
simple_design <- define_design(
  id = "subject",
  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(
  id = "pupil",
  between = list(
    class = list(intervention = c("yes", "no"))
  ),
  nesting_vars = list(class = factor(1:10))
)
```

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, family = "gaussian")
```

Arguments

- | | |
|---------|---|
| formula | The fixed-effects part of the model formula (e.g., $y \sim group * time$). |
| outcome | A data frame containing columns for all predictor variables and exactly one column for the expected outcome values. |
| 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. |

Value

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

Examples

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

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-pasteable 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. <code>outcome ~ predictor1 * predictor2 + (1 id)</code>). Since this function only uses the fixed-effects part of the model, specifying the random effects is optional here. |
| design | A PowR <i>Priori_design</i> object created with <code>define_design()</code> . |

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(
  id = "subject",
  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

<code>formula</code>	An lme4-style model formula (e.g. <code>outcome ~ predictor1 * predictor2 + (1 id)</code>).
<code>design</code>	A PowRPriori_design object created with <code>define_design()</code> .
<code>family</code>	The model family ("gaussian", "binomial", "poisson"). Determines if <code>sd_resid</code> should be included in the template.

Value

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

Examples

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

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

```
plot_sim_model(
  x,
  type = "power_curve",
  design = NULL,
  fixed_effects = NULL,
  random_effects = NULL,
  family = NULL,
  n = NULL,
  x_var = NULL,
  group_var = NULL,
  color_var = NULL,
  facet_var = NULL,
  n_data_points = 10,
  ...
)
```

Arguments

<code>x</code>	The object to plot.
<code>type</code>	The type of plot to create: "power_curve" (default) or "data" (to visualize the sample data from the simulation).
<code>design</code>	A PowRPriori_design object.
<code>fixed_effects, random_effects</code>	Lists of effect parameters.
<code>family</code>	The model family (e.g., "gaussian").
<code>n</code>	The total sample size to simulate for the plot.
<code>x_var, group_var, color_var, facet_var</code>	Strings specifying variables for plot aesthetics.
<code>n_data_points</code>	The maximum number of trajectories in spaghetti plots.
<code>...</code>	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(
  id = "subject",
  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,
  n = 30
)

# 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",
  n_start = 20,
  n_increment = 5,
  n_sims = 100, # Using a smaller n_sims for a quick example
  parallel_plan = "multisession"
)

# 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",
  power_crit = 0.8,
  n_start,
  n_increment,
  max_simulation_steps = 40,
  n_issue_stop_prop = 0.2,
  n_is_total = TRUE,
  n_sims = 2000,
  alpha = 0.05,
  parallel_plan = "multisession"
)
```

Arguments

<code>formula</code>	An lme4-style model formula (e.g. <code>outcome ~ predictor1 * predictor2 + (1 id)</code>).
<code>design</code>	A PowRPriori_design object created by <code>define_design()</code> .
<code>test_parameter</code>	A character vector of the variable names to test for power. If <code>NULL</code> (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 of the variables is aided by the output of the <code>get_fixed_effects_structure()</code> helper function.
<code>fixed_effects</code>	A named list of the fixed-effects coefficients. It is highly recommended to generate this using <code>get_fixed_effects_structure()</code> or <code>fixed_effects_from_average_outcome()</code> .
<code>random_effects</code>	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 <code>get_random_effects_structure()</code> . If this parameter is not used, <code>icc_specs</code> and <code>overall_variance</code> need to be supplied.
<code>icc_specs</code>	Optional. A named list of Intraclass Correlation Coefficients for defining simple random-intercept models. Must be used with <code>overall_variance</code> .
<code>overall_variance</code>	The total variance of the outcome, required when <code>icc_specs</code> is used.
<code>family</code>	The model family: "gaussian" (for LMMs), "binomial" (for logistic GLMMs), or "poisson" (for poisson GLMMs).
<code>power_crit</code>	The desired statistical power level (e.g., 0.80 for 80%).
<code>n_start</code>	The starting sample size for the simulation.

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 40 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_is_total	Boolean that controls how sample sizes are interpreted. If TRUE (default), n_start refers to the total sample size. If FALSE, it refers to the sample size per cell (see define_design() for details on nested designs).
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(
  id = "subject",
  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",
n_start = 20,
n_increment = 5,
n_sims = 100, # Use low n_sims for quick examples
parallel_plan = "multisession"
)

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

```
.create_design_matrix, 2
.plot_data, 3
.simulate_outcome, 4
.to_factor_safely, 4

define_design, 5

fixed_effects_from_average_outcome, 6

get_fixed_effects_structure, 7
get_random_effects_structure, 8

plot_sim_model, 8
power_sim, 11

summary.PowRPriori, 14
```