

# Package ‘powerlmm’

April 17, 2018

**Type** Package

**Title** Power Analysis for Longitudinal Multilevel Models

**Version** 0.3.0

**Description** Calculate power for the 'time x treatment' effect in two- and three-level multilevel longitudinal studies with missing data. Both the third-level factor (e.g. therapists, schools, or physicians), and the second-level factor (e.g. subjects), can be assigned random slopes. Studies with partially nested designs, unequal cluster sizes, unequal allocation to treatment arms, and different dropout patterns per treatment are supported. For all designs power can be calculated both analytically and via simulations. The analytical calculations extends the method described in Galbraith et al. (2002) <doi:10.1016/S0197-2456(02)00205-2>, to three-level models. Additionally, the simulation tools provides flexible ways to investigate bias, Type I errors and the consequences of model misspecification.

**License** GPL (>= 3)

**URL** <https://github.com/rpsychologist/powerlmm>

**BugReports** <https://github.com/rpsychologist/powerlmm/issues>

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.0.1

**Depends** R (>= 3.2.0)

**Imports** stats, methods, parallel, lme4 (>= 1.1), Matrix, MASS, scales, utils

**Suggests** testthat, dplyr, tidyr, knitr, rmarkdown, pbmcapply (>= 1.1), lmerTest (>= 2.0), ggplot2 (>= 2.2), ggsci, viridis, gridExtra, shiny (>= 1.0), shinydashboard

**ByteCompile** true

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Kristoffer Magnusson [aut, cre]

**Maintainer** Kristoffer Magnusson <hello@kristoffer.email>

**Repository** CRAN

**Date/Publication** 2018-04-17 18:28:35 UTC

## R topics documented:

cohend . . . . .	3
create_lmer_formula . . . . .	6
dropout_manual . . . . .	6
dropout_weibull . . . . .	8
get_correlation_matrix . . . . .	10
get_DEFT . . . . .	11
get_dropout . . . . .	12
get_ICC_pre_clusters . . . . .	13
get_ICC_pre_subjects . . . . .	14
get_ICC_slope . . . . .	14
get_monte_carlo_se . . . . .	15
get_power . . . . .	16
get_power_table . . . . .	18
get_sds . . . . .	20
get_slope_diff . . . . .	21
get_var_ratio . . . . .	21
get_VPC . . . . .	22
per_treatment . . . . .	23
plot.plcp . . . . .	24
plot.plcp_ICC2 . . . . .	25
plot.plcp_power_table . . . . .	25
plot.plcp_sds . . . . .	26
plot.plcp_VPC . . . . .	26
powerlmm . . . . .	27
print.plcp_2lvl . . . . .	28
print.plcp_3lvl . . . . .	28
print.plcp_ICC2 . . . . .	29
print.plcp_mc_se . . . . .	29
print.plcp_multi . . . . .	30
print.plcp_multi_power . . . . .	30
print.plcp_multi_sim . . . . .	31
print.plcp_multi_sim_summary . . . . .	31
print.plcp_power_2lvl . . . . .	32
print.plcp_power_3lvl . . . . .	32
print.plcp_sds . . . . .	33
print.plcp_sim . . . . .	33
print.plcp_sim_summary . . . . .	34
print.plcp_VPC . . . . .	34
shiny_powerlmm . . . . .	35
simulate.plcp . . . . .	35

simulate\_data . . . . . 39  
 study\_parameters . . . . . 40  
 summary.plcp\_multi\_sim . . . . . 44  
 summary.plcp\_sim . . . . . 45  
 unequal\_clusters . . . . . 46  
 update.plcp . . . . . 48  
 [.plcp\_multi\_power . . . . . 49

**Index** **50**

cohend *Use Cohen's d as the effect size in study\_parameters*

**Description**

This function is used as input to the effect\_size argument in study\_parameters, if standardized effect sizes should be used. The choice of the denominator differs between fields, and this function supports the common ones: pre- or posttest SD, or the random slope SD.

**Usage**

cohend(ES, standardizer = "pretest\_SD", treatment = "control")

**Arguments**

- ES                    numeric; value of the standardized effect size. Can be a vector.
- standardizer        character; the standardizer (denominator) used to calculate Cohen's d. Allows options are: "pretest\_SD", "posttest\_SD", or "slope\_SD". See Details from more information.
- treatment            character; indicates if the standardizer should be based on the "treatment" or "control" group—this only matters for 3-level partially nested designs.

**Details**

**Standardizing using the pretest\_SD or posttest\_SD**

For these effect sizes, ES indicates the standardized difference between the treatment groups at posttest (T\_end), standardized by using either the implied standard deviation at pretest or posttest. Thus, the actual raw differences in average slopes between the treatments are,

$$\text{slope\_diff} = (\text{ES} * \text{SD}) / \text{T\_end}.$$

**slope\_SD: standardizing using the random slopes**

This standardization is quite different from using the pretest or posttest SD. Here the average slope difference is standardized using the total SD of the random slopes. This is done by e.g. Raudenbush and Liu (2001). **NB**, for this effect size ES indicates the difference in change per unit time, and not at posttest. Thus, the raw difference in average slopes is,

$$\text{slope\_diff} = \text{ES} * \text{slope\_SD}.$$

For a 3-level model, slope\_SD = sqrt(sigma\_subject\_slope^2 + sigma\_cluster\_slope^2).

**Value**

A list of the same length as ES. Each element is a named list of class `plcp_cohend`, with the elements:

- `set`: A helper function that converts the standardized ES to raw values. Accepts a `study_parameters` objects, and returns a numeric indicating the raw difference between the treatment at posttest.
- `get`: contains a list with the original call: "ES", "standardizer", and "treatment".

**References**

Raudenbush, S. W., & Liu, X. F. (2001). Effects of study duration, frequency of observation, and sample size on power in studies of group differences in polynomial change. *Psychological methods*, 6(4), 387.

**See Also**

[study\\_parameters](#)

**Examples**

```
# Pretest SD
p <- study_parameters(n1 = 11,
                    n2 = 20,
                    icc_pre_subject = 0.5,
                    cor_subject = -0.4,
                    var_ratio = 0.03,
                    effect_size = cohend(0.4, standardizer = "pretest_SD"))

get_slope_diff(p)

# using posttest SD,
# due to random slope SD will be larger at posttest
# thus ES = 0.4 indicate larger raw slope difference
# using posttest SD
p <- update(p, effect_size = cohend(0.4,
                                standardizer = "posttest_SD"))

get_slope_diff(p)

# Random slope SD
p <- study_parameters(n1 = 11,
                    n2 = 20,
                    icc_pre_subject = 0.5,
                    cor_subject = -0.4,
                    var_ratio = 0.03,
                    effect_size = cohend(0.4, standardizer = "slope_SD"))

# Partially nested -----
p <- study_parameters(n1 = 11,
                    n2 = 20,
```

```

        n3 = 4,
        icc_pre_subject = 0.5,
        icc_pre_cluster = 0.25,
        cor_subject = -0.4,
        var_ratio = 0.03,
        partially_nested = TRUE,
        effect_size = cohend(0.4, standardizer = "pretest_SD")
    )
# Default is to use control groups SD
get_slope_diff(p)

# Treatment group's SD also include cluster-level intercept variance.
# Thus, ES of 0.4 will indicate a larger raw difference
# using the treatment group's SD
p <- update(p, effect_size = cohend(0.4,
                                   standardizer = "pretest_SD",
                                   treatment = "treatment"))

get_slope_diff(p)

## Combine multiple values, and raw and standardized effects -----
p <- study_parameters(n1 = 11,
                    n2 = 20,
                    icc_pre_subject = 0.5,
                    cor_subject = -0.4,
                    var_ratio = 0.03,
                    effect_size = c(-5, 9,
                                    cohend(c(0.5, 0.8), standardizer = "pretest_SD"),
                                    cohend(c(0.5, 0.8), standardizer = "posttest_SD")))

## Recreate results in Raudenbush & Liu 2001 -----
rauden_liu <- function(D, f, n = 238) {
  n1 <- f * D + 1
  p <- study_parameters(n1 = n1,
                      n2 = n/2,
                      T_end = D,
                      sigma_subject_intercept = sqrt(0.0333),
                      sigma_subject_slope = sqrt(0.0030),
                      sigma_error = sqrt(0.0262),
                      effect_size = cohend(0.4, standardizer = "slope_SD"))

  x <- get_power(p)
  round(x$power, 2)
}

## Table 1 in Raudenbush & Liu 2001
## NB, it looks like they made an error in column 1.
g <- expand_grid(D = 2:8,
                f = c(0.5, 1:6))
g$power <- mapply(rauden_liu, D = g$D, f = g$f)
tidyr::spread(g, f, power)

## Table 3 Table 1 in Raudenbush & Liu 2001

```

```
g <- expand.grid(n = seq(100, 800, by = 100),
               D = 4,
               f = c(0.5, 1:6))
g$power <- mapply(rauden_liu, n = g$n, f = g$f, D = g$D)
tidyr::spread(g, n, power)
```

---

create\_lmer\_formula    *Create an lmer formula based on a [study\\_parameters-object](#)*

---

### Description

Create an lmer formula based on a [study\\_parameters-object](#)

### Usage

```
create_lmer_formula(object, ...)
```

### Arguments

object            A [study\\_parameters-object](#) containing one study design  
 ...                Unused, optional arguments.

### Details

The lme4 formula will correspond to the model implied by the specified parameters in the [study\\_parameters-object](#). Thus, if e.g. `cor_subject` is NA or NULL the corresponding term is removed from the lmer formula. Parameters that are 0 are retained.

Currently only objects with one study design are supported, i.e. objects with class `plcp`, and not `plcp_multi`; `data.frame` with multiple designs are currently not supported.

### Value

A character vector with lmer formula syntax.

---

dropout\_manual        *Manually specify dropout per time point*

---

### Description

Used as input to the dropout-argument in [study\\_parameters](#).

### Usage

```
dropout_manual(...)
```

## Arguments

... The proportion of dropout per time point, either as a vector of length  $n1$ , or  $n1$  individual numeric arguments, see *Details*.

## Details

Specifying dropout manually requires that the dropout is 0 at the first time point. Moreover, dropout can't decrease over time and can never be 1.

## Value

A list of class `plcp_dropout_manual`

## See Also

[dropout\\_weibull](#), [per\\_treatment](#)

## Examples

```
dropout <- dropout_manual(0, 0, 0, 0, 0.2, 0.2, 0.3, 0.3, 0.4, 0.4, 0.45)
```

```
p <- study_parameters(n1 = 11,
                    n2 = 5,
                    n3 = 6,
                    T_end = 10,
                    icc_pre_subject = 0.5,
                    icc_pre_cluster = 0,
                    var_ratio = 0.03,
                    icc_slope = 0.05,
                    dropout = dropout,
                    cohend = -0.8)
```

```
plot(p, plot = 2)
get_power(p)
```

```
# Can also use a vector as input
```

```
dropout <- dropout_manual(seq(0, 0.5, length.out = 11))
```

```
p <- study_parameters(n1 = 11,
                    n2 = 5,
                    n3 = 6,
                    T_end = 10,
                    icc_pre_subject = 0.5,
                    icc_pre_cluster = 0,
                    var_ratio = 0.03,
                    icc_slope = 0.05,
                    dropout = dropout,
                    cohend = -0.8)
```

```
plot(p, plot = 2)
get_power(p)
```

```
## Not run:
```

```
# Decreasing dropout will throw an error
```

```
dropout_manual(0, 0.1, 0.1, 0.2, 0.1)
```

```
# Dropout at the first time point will throw an error
dropout_manual(0.1, 0.1, 0.1, 0.2, 0.2)

## End(Not run)
```

---

dropout\_weibull      *Use the Weibull distribution to specify the dropout process*

---

### Description

Used as input to the dropout-argument in [study\\_parameters](#)

### Usage

```
dropout_weibull(proportion, rate)
```

### Arguments

proportion	Total proportion of subjects that have dropped out at the last time point. Must be less than 1.
rate	Indicates the "shape" of the dropout process, if $> 1$ then dropout is concentrated at the end of the study, if $rate < 1$ more dropout occurs at the beginning of the study. If $rate == 1$ the risk of dropout is constant.

### Details

N.B a constant ( $rate = 1$ ) hazard of dropout does not mean dropout is linear over time. It means that the risk of dropping out at the next time point is constant over the study period.

### Value

A `plcp_weibull` named list, with the first element containing the dropout function.

### References

Galbraith, S., Stat, M., & Marschner, I. C. (2002). Guidelines for the design of clinical trials with longitudinal outcomes. *Controlled clinical trials*, 23(3), 257-273.

### See Also

[dropout\\_manual](#), [per\\_treatment](#)



**Examples**

```
p <- study_parameters(n1 = 11,
                    n2 = 5,
                    n3 = 6,
                    T_end = 10,
                    icc_pre_subject = 0.5,
                    icc_pre_cluster = 0,
                    var_ratio = 0.03,
                    icc_slope = 0.05,
                    dropout = dropout_weibull(proportion = 0.3, rate = 3),
                    cohend = -0.8)
```

```
get_dropout(p)
plot(p, plot = 2)
```

```
# Different per treatment
tx <- dropout_weibull(proportion = 0.3, rate = 3)
cc <- dropout_weibull(proportion = 0.3, rate = 1/3)
dropout <- per_treatment(control = cc,
                       treatment = tx)
```

```
p <- study_parameters(n1 = 11,
                    n2 = 5,
                    n3 = 6,
                    T_end = 10,
                    icc_pre_subject = 0.5,
                    icc_pre_cluster = 0,
                    var_ratio = 0.03,
                    icc_slope = 0.05,
                    dropout = dropout,
                    cohend = -0.8)
```

```
plot(p, plot = 2)
```

```
# Compare power for different dropout amounts
dropout <- c(dropout_weibull(proportion = 0.3, rate = 3),
             dropout_weibull(proportion = 0.5, rate = 3),
             dropout_weibull(proportion = 0.5, rate = 1/3))
```

```
p <- study_parameters(n1 = 11,
                    n2 = 5,
                    n3 = 6,
                    T_end = 10,
                    icc_pre_subject = 0.5,
                    icc_pre_cluster = 0,
                    var_ratio = 0.03,
                    icc_slope = 0.05,
                    dropout = dropout,
                    cohend = -0.8)
```

```
get_power(p)
```

---

`get_correlation_matrix`*Calculate the subject-level (ICC) correlations among time points*

---

### Description

Calculate the subject-level (ICC) correlations among time points

### Usage

```
get_correlation_matrix(object)

## S3 method for class 'plcp_multi'
get_correlation_matrix(object)
```

### Arguments

`object` An object created by [study\\_parameters](#)

### Details

The correlation between time point  $T_i$  and  $T_{i+1}$  within the same subject is also called the intraclass correlation (ICC) at level two. If the random slopes are non-zero this ICC change over time.

### Value

A  $n_1 \times n_1$  matrix with the marginal subject-level correlations between time points.

### Examples

```
paras <- study_parameters(n1 = 11,
                        n2 = 10,
                        n3 = 3,
                        T_end = 10,
                        icc_pre_subject = 0.5,
                        icc_pre_cluster = 0,
                        icc_slope = 0.05,
                        var_ratio = 0.03)
get_correlation_matrix(paras)
```

---

`get_DEFT`*Calculate the design effect and Type I errors*

---

### Description

This functions helps to evaluate the consequences of ignoring a random slope at the cluster level.

### Usage

```
get_DEFT(object)

## S3 method for class 'plcp_3lvl'
get_DEFT(object)
```

### Arguments

`object` A `plcp_3lvl`-object created by [study\\_parameters](#)

### Details

The design effect (DEFT) is the ratio of the standard error from the correct three-level model to the standard error from the misspecified model omitting the cluster-level random slope. The standard error for the misspecified model is calculated by assuming that the cluster-level random slope variance is added to the subject-level random slope.

The approximate Type I error under the miss-specified model is also calculated. The effect of wrongly ignoring a third-level random slope on the Type I errors, depends on `n1`, `n2`, `n3`, `icc_slope`, and, `var_ratio`.

### Value

A data.frame with the columns `n1`, `n2`, `n3`, `icc_slope`, `var_ratio`, `DEFT`, and, `approx_type1`. The number of rows of the data.frame will be equals to the number of different combination of parameters values specified with [study\\_parameters](#).

### See Also

[simulate.plcp](#)

### Examples

```
paras <- study_parameters(n1 = 11,
                        n2 = 30,
                        n3 = 3,
                        T_end = 10,
                        icc_pre_subject = 0.5,
                        icc_pre_cluster = 0,
                        icc_slope = c(0.01,0.05, 0.1),
                        var_ratio = 0.02)
```



---

get\_ICC\_pre\_clusters *Calculate the amount of baseline variance at the cluster level*

---

### Description

Calculate the amount of baseline variance at the cluster level

### Usage

```
get_ICC_pre_clusters(object, ...)
```

### Arguments

object	An object created by <a href="#">study_parameters</a>
...	Optional named arguments.

### Details

The proportion of variance at the cluster level at baseline can be interpreted as the correlation between two subjects belonging to the same cluster.

### Value

Returns the proportion of baseline variance at the cluster level, as a numeric vector.

### Examples

```
paras <- study_parameters(n1 = 11,  
                        n2 = 10,  
                        n3 = 3,  
                        T_end = 10,  
                        sigma_subject_intercept = 1.2,  
                        sigma_subject_slope = 0.2,  
                        sigma_cluster_intercept = 0.5,  
                        sigma_cluster_slope = 0.2,  
                        sigma_error = 1.2,  
                        cohend = -0.8)  
  
get_ICC_pre_clusters(paras)
```

---

get\_ICC\_pre\_subjects    *Calculate the subject-level ICC at pretest*

---

**Description**

Calculate the subject-level ICC at pretest

**Usage**

```
get_ICC_pre_subjects(object, ...)
```

**Arguments**

object            An object created by [study\\_parameters](#)  
...                Optional named arguments.

**Value**

Returns the proportion of baseline variance at the subject level (which also includes cluster-level variance), as a numeric vector.

**Examples**

```
paras <- study_parameters(n1 = 11,  
                          n2 = 10,  
                          n3 = 3,  
                          T_end = 10,  
                          sigma_subject_intercept = 1.2,  
                          sigma_subject_slope = 0.2,  
                          sigma_cluster_intercept = 0.5,  
                          sigma_cluster_slope = 0.2,  
                          sigma_error = 1.2,  
                          cohend = -0.8)  
  
get_ICC_pre_subjects(paras)
```

---

get\_ICC\_slope            *Calculate the amount of slope variance at the third level*

---

**Description**

Calculate the amount of slope variance at the third level

**Usage**

```
get_ICC_slope(object, ...)
```

**Arguments**

object            An object created by [study\\_parameters](#).  
 ...                Optional named arguments.

**Value**

Returns the proportion of slope variance at the third level as a numeric vector. NA is returned for models with no slope variance as either level two or three.

**Examples**

```
paras <- study_parameters(n1 = 11,
                          n2 = 10,
                          n3 = 3,
                          T_end = 10,
                          sigma_subject_intercept = 1.2,
                          sigma_subject_slope = 0.2,
                          sigma_cluster_intercept = 0,
                          sigma_cluster_slope = 0.2,
                          sigma_error = 1.2,
                          cohend = -0.8)
```

```
get_ICC_slope(paras)
```

---

get\_monte\_carlo\_se        *Calculate the Monte Carlo standard error of the empirical power estimates*

---

**Description**

Returns the expected simulation error for a study design. Indicates how many simulation that are needed for a desired precision in the empirical power estimates.

**Usage**

```
get_monte_carlo_se(object, nsim, power, ...)
```

```
## S3 method for class 'plcp_power_3lv1'
get_monte_carlo_se(object, nsim, ...)
```

```
## S3 method for class 'plcp_power_2lv1'
get_monte_carlo_se(object, nsim, ...)
```

**Arguments**

object            An object created by [get\\_power](#)  
 nsim              A numeric indicating the number of simulations  
 power             *Optional*. A numeric indicating the empirical power.  
 ...                Currently not used. Used when object is NULL.

**Value**

A data.frame with the estimated power, expected standard error of the simulated power estimate, and the 95 % CI of the estimate.

**Examples**

```
paras <- study_parameters(n1 = 11,
                        n2 = 10,
                        n3 = 6,
                        T_end = 10,
                        icc_pre_subject = 0.5,
                        icc_pre_cluster = 0,
                        var_ratio = 0.03,
                        icc_slope = 0.05,
                        cohend = -0.8)

x <- get_power(paras)
get_monte_carlo_se(x, nsim = 1000)

# Without an object
get_monte_carlo_se(power = 0.8, nsim = 1000)
```

---

get\_power

---

*Calculate power for two- and three-level models with missing data.*


---

**Description**

Calculate power for two- and three-level models with missing data.

**Usage**

```
get_power(object, df = "between", alpha = 0.05, progress = TRUE, R = 1L,
          cores = 1L, ...)
```

**Arguments**

object	An object created by <a href="#">study_parameters</a>
df	Either "between" or "satterth" for Satterthwaite's DF approximation. Also accepts a numeric value which will be used as DF.
alpha	The alpha level, defaults to 0.05.
progress	logical; displays a progress bar when > 1 power analysis is performed.
R	An integer indicating how many realizations to base power on. Useful when dropout or cluster sizes are sampled (i.e. are random variables).
cores	An integer indicating how many CPU cores to use.
...	Other potential arguments; currently used to pass progress bar from Shiny





```

# With missing data
paras <- study_parameters(n1 = 11,
                        n2 = 40,
                        T_end = 10,
                        icc_pre_subject = 0.5,
                        var_ratio = 0.02,
                        dropout = dropout_weibull(0.3, 2),
                        cohend = -0.8)

get_power(paras)

# Three-level model
paras <- study_parameters(n1 = 11,
                        n2 = 10,
                        n3 = 5,
                        T_end = 10,
                        icc_pre_subject = 0.5,
                        icc_pre_cluster = 0,
                        icc_slope = 0.05,
                        var_ratio = 0.02,
                        cohend = -0.8)

get_power(paras)

# With missing data
paras <- study_parameters(n1 = 11,
                        n2 = 10,
                        n3 = 5,
                        T_end = 10,
                        icc_pre_subject = 0.5,
                        icc_pre_cluster = 0,
                        icc_slope = 0.05,
                        var_ratio = 0.02,
                        dropout = dropout_weibull(0.3, 2),
                        cohend = -0.8)

get_power(paras)

# Satterthwaite DFs
get_power(paras, df = "satterthwaite")

```

---

get\_power\_table

---

*Create a power table for a combination of parameter values*


---

### Description

Create a power table for a combination of parameter values

**Usage**

```
get_power_table(object, n2, ..., df = "between", alpha = 0.05, R = 1L,
  cores = 1L)
```

**Arguments**

object	An object created by <a href="#">study_parameters</a>
n2	A vector of n2 values
...	Optional named arguments. Up to two extra arguments can be compared. When used together with the plot method, the first argument will be grouped by color and the second by facets.
df	Either "between" or "satterth" for Satterthwaite's DF approximation. Also accepts a numeric value which will be used as DF. See <a href="#">get_power</a>
alpha	The alpha level, defaults to 0.05.
R	An integer indicating how many realizations to base power on. Useful when dropout or cluster sizes are sampled (i.e. are random variables).
cores	An integer indicating how many CPU cores to use.

**Value**

A data.frame with class plcp\_power\_table.

**Examples**

```
paras <- study_parameters(n1 = 11,
  n2 = 10,
  n3 = 6,
  T_end = 10,
  icc_pre_subject = 0.5,
  icc_pre_cluster = 0,
  var_ratio = 0.03,
  icc_slope = 0.05,
  cohend = -0.8)

# increase only n2
x <- get_power_table(paras, n2 = 10:15)
plot(x)

# Compare two parameters
x <- get_power_table(paras, n2 = 10:15, n3 = 6:8)
plot(x)

# Compare impact of three parameters
x <- get_power_table(paras, n2 = seq(3, 25, by = 3),
  n3 = c(3,6,9),
  icc_slope = c(0, 0.05, 0.1))

plot(x)
```

---

`get_sds`*Calculate the model implied standard deviations per time point*

---

**Description**

Calculate the model implied standard deviations per time point

**Usage**

```
get_sds(object, treatment = "treatment", n = 1)
```

**Arguments**

<code>object</code>	An object created by <a href="#">study_parameters</a>
<code>treatment</code>	character; either "treatment" or "control". Indicates for which group SDs should be calculated for. This only makes a difference for 3-level partially nested designs.
<code>n</code>	Optional; selects row n if object is a data.frame of parameters

**Value**

data.frame with class `plcp_sds` containing the model implied standard deviations per time point.

**See Also**

[get\\_VPC](#), [get\\_correlation\\_matrix](#)

**Examples**

```
paras <- study_parameters(n1 = 11,
                        n2 = 10,
                        n3 = 6,
                        T_end = 10,
                        icc_pre_subject = 0.5,
                        icc_pre_cluster = 0,
                        icc_slope = 0.05,
                        var_ratio = 0.03)

get_sds(paras)

# plot
plot(get_sds(paras))
```

---

get_slope_diff	<i>Return the raw difference between the groups at posttest</i>
----------------	---

---

**Description**

Used internally to calculate the difference in change over time between the two treatment groups.

**Usage**

```
get_slope_diff(object)

## S3 method for class 'plcp'
get_slope_diff(object)

## S3 method for class 'plcp_multi'
get_slope_diff(object)
```

**Arguments**

object            A [study\\_parameters](#)-object.

**Value**

A numeric indicating the mean difference between the treatment and control group at posttest.

---

get_var_ratio	<i>Calculates the ratio of the slope variance to the within-subjects error variance</i>
---------------	---

---

**Description**

Calculates the ratio of the slope variance to the within-subjects error variance

**Usage**

```
get_var_ratio(object, ...)
```

**Arguments**

object            An object created by [study\\_parameters](#)  
 ...                Optional arguments.

**Value**

Returns the ratio of the total slope variance to the within-subject error as a numeric vector.

**Examples**

```

paras <- study_parameters(n1 = 11,
                          n2 = 10,
                          n3 = 3,
                          T_end = 10,
                          sigma_subject_intercept = 1.2,
                          sigma_subject_slope = 0.2,
                          sigma_cluster_intercept = 0,
                          sigma_cluster_slope = 0.2,
                          sigma_error = 1.2,
                          cohend = -0.8)

get_var_ratio(paras)

```

---

get\_VPC

---

*Calculate the variance partitioning coefficient*


---

**Description**

Calculate the variance partitioning coefficient

**Usage**

```

get_VPC(object)

## S3 method for class 'plcp'
get_VPC(object)

```

**Arguments**

object            An object created by [study\\_parameters](#)

**Details**

For partially nested studies, the VPC is calculated for the treatment group.

**Value**

a data.frame with class plcp\_VPC containing the percentage of variance per level and time point. The column `between_clusters` is also the intraclass correlation for level three, i.e. the correlation between two subjects belonging to the same cluster at a specific time point. With random slopes in the model the variances per time point will be a quadratic function of time. `tot_var` is the percentage increase or decrease in total variance relative to baseline variance.

The `plot` method returns a `ggplot2::ggplot` object.

## References

Goldstein, H., Browne, W., & Rasbash, J. (2002). Partitioning variation in multilevel models. *Understanding Statistics: Statistical Issues in Psychology, Education, and the Social Sciences*, 1(4), 223-231.

## See Also

[plot.plcp\\_VPC](#)

## Examples

```
paras <- study_parameters(n1 = 11,
                        n2 = 10,
                        n3 = 3,
                        T_end = 10,
                        icc_pre_subject = 0.5,
                        icc_pre_cluster = 0,
                        icc_slope = 0.05,
                        var_ratio = 0.03)

res <- get_VPC(paras)
res

# Plot
plot(res)
```

---

per\_treatment

*Setup parameters that differ per treatment group*

---

## Description

Helps specifying unequal cluster sizes with [study\\_parameters](#), e.g. different number of clusters in the treatment and control arm, or different dropout patterns.

## Usage

```
per_treatment(control, treatment)
```

## Arguments

control	Value used for control group
treatment	Value used for treatment group

## Details

The type of object passed to control and treatment will depend on the parameters in [study\\_parameters](#) that should have different values per treatment group.

**Value**

An object of class "plcp\_per\_treatment"

**See Also**

[unequal\\_clusters](#), [study\\_parameters](#), [dropout\\_weibull](#)

**Examples**

```
n2 <- per_treatment(control = 10,
                    treatment = 20)
p <- study_parameters(n1 = 11,
                     n2 = n2,
                     n3 = 6,
                     T_end = 10,
                     icc_pre_subject = 0.5,
                     icc_pre_cluster = 0,
                     var_ratio = 0.03,
                     icc_slope = 0.05,
                     cohend = -0.8)
```

---

plot.plcp

*Plot method for study\_parameters-objects*

---

**Description**

Plot method for study\_parameters-objects

**Usage**

```
## S3 method for class 'plcp'
plot(x, n = 1, type = "both", ...)
```

**Arguments**

x	An object of class plcp.
n	specifies which row n should be used if object is a data.frame containing multiple setups.
type	indicated what plot to show. If effect the plot showing the treatment groups change over time will be shown, if dropout the missing data pattern will be shown, if both both plots will be shown.
...	Optional arguments.



---

plot.plcp\_ICC2      *Plot method for get\_correlation\_matrix-objects*

---

**Description**

Plot method for get\_correlation\_matrix-objects

**Usage**

```
## S3 method for class 'plcp_ICC2'  
plot(x, ...)
```

**Arguments**

x                    An object created with [get\\_correlation\\_matrix](#)  
...                   Optional arguments, currently ignored.

---

plot.plcp\_power\_table      *Plot method for get\_power\_table-objects*

---

**Description**

Plot method for get\_power\_table-objects

**Usage**

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

**Arguments**

x                    An object of class plcp\_power\_table.  
...                   Optional arguments.

---

plot.plcp_sds	<i>Plot method for get_sds-objects</i>
---------------	--

---

**Description**

Plot method for get\_sds-objects

**Usage**

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

**Arguments**

x	An object of class plcp_sds.
...	Optional arguments.

---

plot.plcp_VPC	<i>Plot method for get_VPC-objects</i>
---------------	--

---

**Description**

Plot method for get\_VPC-objects

**Usage**

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

**Arguments**

x	An object created with <a href="#">get_VPC</a>
...	Optional arguments, currently ignored.

## Description

The **powerlmm** package provides a fast and flexible way to calculate power for two- and three-level multilevel models with missing data. The focus is on power analysis for the test of the treatment effect in longitudinally clustered designs, i.e. where the first level is measurements, and the second level is subjects nested within a (optional) higher level-three unit, e.g. therapists.

## Details

All study designs are specified using the function `study_parameters`, which lets you define your model using familiar notation, either by specifying the model parameters directly, or by using relative standardized inputs (e.g. % variance at each level). Several functions are provided to help you visualize and understand the implied model, type `methods(class="plcp")` to see available methods. The basic features of the package are also available via an interactive (**Shiny**) web application, which you can launch by typing `shiny_powerlmm()`.

## Supported models

The purpose of **powerlmm** is to help design longitudinal treatment studies, with or without higher-level clustering (e.g. by therapists, groups, or physicians), and missing data. The main features of the package are:

- Longitudinal Two- and three-level (nested) linear mixed models, and partially nested designs
- Random slopes at the subject- and cluster-level.
- Account for missing data/dropout.
- Unbalanced designs (both unequal cluster sizes, and treatment groups).
- Calculate the design effect, and estimated type I error when the third-level is ignored.
- Fast analytical power calculations for all supported designs.
- Explore bias, Type I error and model misspecification using convenient simulation methods
- Few clusters; accurate power analysis even with very few clusters, by using Satterthwaite's degrees of freedom approximation.
- Create power curves to guide power analysis and help with optimal design of sample sizes at each level.

## Tutorials

Type `vignette("two-level", package = "powerlmm")`, or `vignette("three-level", package = "powerlmm")` to see a tutorial on using **powerlmm** to calculate power. See all available vignettes by typing `vignette(package = "powerlmm")`.

**Author(s)**

Kristoffer Magnusson

Maintainer: Kristoffer Magnusson <hello@kristoffer.email>

**See Also**

[study\\_parameters](#), [get\\_power](#)

---

print.plcp\_2lv1      *Print method for two-level study\_parameters-objects*

---

**Description**

Print method for two-level study\_parameters-objects

**Usage**

```
## S3 method for class 'plcp_2lv1'  
print(x, ...)
```

**Arguments**

x	An object of class plcp_2lv1.
...	Optional arguments.

---

print.plcp\_3lv1      *Print method for three-level study\_parameters-objects*

---

**Description**

Print method for three-level study\_parameters-objects

**Usage**

```
## S3 method for class 'plcp_3lv1'  
print(x, ...)
```

**Arguments**

x	An object of class plcp_3lv1.
...	Optional arguments.

---

print.plcp\_ICC2      *Print method for get\_correlation\_matrix-objects*

---

**Description**

Print method for get\_correlation\_matrix-objects

**Usage**

```
## S3 method for class 'plcp_ICC2'  
print(x, ...)
```

**Arguments**

x	An object created by <a href="#">get_correlation_matrix</a>
...	Optional arguments

---

print.plcp\_mc\_se      *Print method for get\_monte\_carlo\_se-objects*

---

**Description**

Print method for get\_monte\_carlo\_se-objects

**Usage**

```
## S3 method for class 'plcp_mc_se'  
print(x, digits = 2, ...)
```

**Arguments**

x	An object created with <a href="#">get_monte_carlo_se</a> .
digits	The number of digits to print.
...	Optional arguments.

print.plcp\_multi      *Print method for study\_parameters-multiobjects*

---

**Description**

Print method for study\_parameters-multiobjects

**Usage**

```
## S3 method for class 'plcp_multi'  
print(x, print_max = 10, empty = ".", digits = 2,  
      ...)
```

**Arguments**

x	An object of class plcp_multi.
print_max	The number of rows to show
empty	Symbol used to replace repeating non-unique parameters
digits	Digits to show
...	Optional arguments.

---

print.plcp\_multi\_power      *Print method for get\_power-multi*

---

**Description**

Print method for get\_power-multi

**Usage**

```
## S3 method for class 'plcp_multi_power'  
print(x, ...)
```

**Arguments**

x	An object of class plcp_multi_power.
...	Optional arguments

---

`print.plcp_multi_sim` *Print method for simulate.plcp\_multi-objects*

---

### **Description**

Print method for `simulate.plcp_multi-objects`

### **Usage**

```
## S3 method for class 'plcp_multi_sim'  
print(x, ...)
```

### **Arguments**

`x` An object created with `simulate.plcp_multi`  
`...` Optional arguments.

---

`print.plcp_multi_sim_summary`  
*Print method for summary.plcp\_multi\_sim-objects*

---

### **Description**

Print method for `summary.plcp_multi_sim-objects`

### **Usage**

```
## S3 method for class 'plcp_multi_sim_summary'  
print(x, digits = 2, ...)
```

### **Arguments**

`x` An object of class `plcp_multi_sim_summary`  
`digits` number of significant digits.  
`...` Optional arguments.

---

print.plcp\_power\_2lvl *Print method for two-level get\_power*

---

**Description**

Print method for two-level get\_power

**Usage**

```
## S3 method for class 'plcp_power_2lvl'  
print(x, ...)
```

**Arguments**

x	An object of class plcp_power_2lvl.
...	Optional arguments

---

print.plcp\_power\_3lvl *Print method for three-level get\_power*

---

**Description**

Print method for three-level get\_power

**Usage**

```
## S3 method for class 'plcp_power_3lvl'  
print(x, ...)
```

**Arguments**

x	An object of class plcp_power_3lvl.
...	Optional arguments



---

print.plcp\_sds      *Print method for get\_sds-objects*

---

**Description**

Print method for get\_sds-objects

**Usage**

```
## S3 method for class 'plcp_sds'  
print(x, ...)
```

**Arguments**

x                    An object of class plcp\_sds.  
...                   Optional arguments.

---

print.plcp\_sim      *Print method for simulate.plcp-objects*

---

**Description**

Print method for simulate.plcp-objects

**Usage**

```
## S3 method for class 'plcp_sim'  
print(x, ...)
```

**Arguments**

x                    An object created with [simulate.plcp](#)  
...                   Optional arguments.

---

```
print.plcp_sim_summary
```

*Print method for summary.plcp\_sim-objects*

---

### Description

Print method for summary.plcp\_sim-objects

### Usage

```
## S3 method for class 'plcp_sim_summary'  
print(x, digits = 2, ...)
```

### Arguments

x	An object of class plcp_sim_summary
digits	number of significant digits.
...	Optional arguments.

---

```
print.plcp_VPC
```

*Print method for get\_vpc-objects*

---

### Description

Print method for get\_vpc-objects

### Usage

```
## S3 method for class 'plcp_VPC'  
print(x, digits = 2, ...)
```

### Arguments

x	Object created with link{get_VPC}
digits	Number of digits to print
...	Optional arguments

---

shiny_powerlmm	<i>Launch powerlmm's Shiny web application</i>
----------------	--

---

### Description

This Shiny application provides the basic functionality of the **powerlmm**-package in a user friendly web application.

### Usage

```
shiny_powerlmm()
```

### Examples

```
## Not run:  
  
library(shiny)  
shiny_powerlmm()  
  
## End(Not run)
```

---

simulate.plcp	<i>Perform a simulation study using a study_parameters-object</i>
---------------	---

---

### Description

Perform a simulation study using a `study_parameters-object`

### Usage

```
## S3 method for class 'plcp'  
simulate(object, nsim, seed = NULL, formula = NULL,  
         satterthwaite = FALSE, CI = FALSE, cores = 1, progress = FALSE,  
         batch_progress = TRUE, ...)  
  
## S3 method for class 'plcp_multi'  
simulate(object, nsim, seed = NULL, formula = NULL,  
         satterthwaite = FALSE, CI = FALSE, cores = 1, progress = FALSE,  
         batch_progress = TRUE, ...)
```

## Arguments

object	An object created by <a href="#">study_parameters</a> .
nsim	The number of simulations to run.
seed	Currently ignored.
formula	<code>lme4::lmer</code> formula(s) used to analyze the data, see <i>Details</i> . Should either be a character vector or named list if one model is simulated. It also possible to compare two models, e.g. a correct and a misspecified model, by passing the two formulas as a named list, with the names "wrong" and "correct". See <i>Examples</i> . If NULL the formula is made automatically, using <a href="#">create_lmer_formula</a> , which does not support objects with multiple simulation setups.
satterthwaite	Logical; if TRUE Satterthwaite's degrees of freedom approximation will be used when computing <i>p</i> -values. This is implemented using the <code>lmerTest</code> -package. See <i>Details</i> .
CI	Logical; if TRUE coverage rates for confidence intervals will be calculated. See <i>Details</i> .
cores	Number of CPU cores to use. If called from a GUI environment (e.g. RStudio) or a computer running Microsoft Windows, PSOCK clusters will be used. If called from a non-interactive Unix environment forking is utilized.
progress	logical; will display progress if TRUE. Currently ignored on <i>Windows</i> . Package <code>pbmclapply</code> is used to display progress, which relies on forking. <b>N.B</b> using a progress bar will noticeably increase the simulation time, due to the added overhead.
batch_progress	logical; if TRUE progress will be shown for simulations with multiple setups.
...	Optional arguments, see <i>Saving</i> in <i>Details</i> section.

## Details

### Model formula

The available model terms are:

- `y` the outcome vector, with potential missing data.
- `y_c` the complete version of `y`, before dropout was simulated.
- `time` the time vector.
- `treatment` treatment indicator (0 = "control", 1 = "treatment").
- `subject` subject-level id variable, from 1 to total number of subjects.
- `cluster` for three-level models; the cluster-level id variable, from 1 to the total number of clusters.

Currently, the models are assumed to be of the form `"y ~ time * treatment + (time | subject) + (time | clusters)"` i.e., "subject" and "cluster" are included as random effects. The `summary.plcp_sim` method will not work if you include parameters that are not defined by the `study_parameters`-object, e.g. `"y ~ time * cluster + (1 | subject)"`.

See *Examples* and the simulation-vignette for formula examples. For objects that contain a single study setup, then the `lmer` formula can be created automatically using [create\\_lmer\\_formula](#).

### Satterthwaite's approximation, and CI coverage

To decrease the simulation time Satterthwaite's *dfs* and the CIs' coverage rates will only be calculated for the test of 'time:treatment'-interaction.

Confidence intervals are both calculated using profile likelihood and by the Wald approximation.

### Saving intermediate results for multi-sims

Objects with multi-sims can be save after each batch is finished. This is highly recommended when many designs are simulated. The following additional arguments control saving behavior:

- 'save', logical, if TRUE each batch is saved as a RDS-file. Results are saved in your working directory, in the directory specified by `save_folder`.
- 'save\_folder' a character indicating the folder name. Default is 'save'.
- 'save\_folder\_create', logical, if TRUE then `save_folder` will be created if it does not exist in your working directory.

### See Also

[summary.plcp\\_sim](#), [simulate\\_data](#)

### Examples

```
## Not run:
# Two-level -----
p <- study_parameters(n1 = 11,
                    n2 = 25,
                    sigma_subject_intercept = 1.44,
                    sigma_subject_slope = 0.2,
                    sigma_error = 1.44,
                    effect_size = cohend(0.5))

f <- "y ~ treatment * time + (1 + time | subject)"

res <- simulate(object = p,
              nsim = 1000,
              formula = f,
              satterthwaite = TRUE,
              progress = FALSE,
              cores = 1,
              save = FALSE)

summary(res)

# Three-level (nested) -----
p <- study_parameters(n1 = 10,
                    n2 = 20,
                    n3 = 4,
                    sigma_subject_intercept = 1.44,
                    icc_pre_cluster = 0,
                    sigma_subject_slope = 0.2,
```

```

        icc_slope = 0.05,
        sigma_error = 1.44,
        effect_size = cohend(0.5))

## compare correct and miss-specified model
f <- list("correct" = "y ~ treatment * time + (1 + time | subject) + (time | cluster)",
        "wrong" = "y ~ treatment * time + (1 + time | subject)")

res <- simulate(object = p,
               nsim = 1000,
               formula = f,
               satterthwaite = TRUE,
               progress = FALSE,
               cores = 1,
               save = FALSE)

summary(res)

# Partially nested design -----
p <- study_parameters(n1 = 11,
                    n2 = 10,
                    n3 = 4,
                    sigma_subject_intercept = 1.44,
                    icc_pre_cluster = 0,
                    sigma_subject_slope = 0.2,
                    cor_subject = -0.5,
                    icc_slope = 0.05,
                    sigma_error = 1.44,
                    partially_nested = TRUE,
                    effect_size = cohend(-0.5))

f <- "y ~ treatment * time + (1 + time | subject) + (0 + treatment:time | cluster)"

res <- simulate(object = p,
               nsim = 1000,
               formula = f,
               satterthwaite = TRUE,
               progress = FALSE,
               cores = 4,
               save = FALSE)

summary(res)

# Run multiple designs -----
p <- study_parameters(n1 = 10,
                    n2 = 20,
                    n3 = c(2, 4, 6),
                    sigma_subject_intercept = 1.44,
                    icc_pre_cluster = 0,
                    sigma_subject_slope = 0.2,
                    icc_slope = 0.05,
                    sigma_error = 1.44,

```

```

effect_size = cohend(0.5))

f <- list("correct" = "y ~ treatment * time + (1 + time | subject) + (time | cluster)",
         "wrong" = "y ~ treatment * time + (1 + time | subject)")

res <- simulate(object = p,
               nsim = 1000,
               formula = f,
               satterthwaite = TRUE,
               progress = FALSE,
               cores = 1,
               save = FALSE)

# Summarize 'time:treatment' results for n3 = c(2, 4, 6) for 'correct' model
summary(res, para = "time:treatment", type = "fixed", model = "correct")

# Summarize cluster-level random slope for n3 = c(2, 4, 6) for 'correct' model
summary(res, para = "cluster_slope", type = "random", model = "correct")

## End(Not run)

```

---

simulate\_data

*Generate a data set using a study\_parameters-object*


---

## Description

Generate a data set using a study\_parameters-object

## Usage

```

simulate_data(paras, n = 1)

## S3 method for class 'plcp'
simulate_data(paras, n = NULL)

## S3 method for class 'plcp_multi'
simulate_data(paras, n = 1)

```

## Arguments

paras	An object created by <a href="#">study_parameters</a>
n	Optional; specifies which row n should be used if object is a data.frame containing multiple setups.

## Value

A data.frame with the simulated data in long form. With the following columns:

- y the outcome vector, with missing values as NA

- `y_c` the outcome vector, without missing values removed.
- `time` the time vector
- `treatment` treatment indicator (0 = "control", 1 = "treatment")
- `subject` subject-level id variable, from 1 to total number of subjects.
- `cluster` for three-level models; the cluster-level id variable, from 1 to the total number of clusters.

### Examples

```
p <- study_parameters(n1 = 11,
                     n2 = 10,
                     n3 = 4,
                     T_end = 10,
                     fixed_intercept = 37,
                     fixed_slope = -0.65,
                     sigma_subject_intercept = 2.89,
                     sigma_cluster_intercept = 0.6,
                     icc_slope = 0.1,
                     var_ratio = 0.03,
                     sigma_error = 1.5,
                     cor_subject = -0.5,
                     cor_cluster = 0,
                     cohend = 0.5)

d <- simulate_data(p)
```

---

study_parameters	<i>Setup study parameters</i>
------------------	-------------------------------

---

### Description

Setup the parameters for calculating power for longitudinal multilevel studies comparing two groups. Ordinary two-level models (subjects with repeated measures), and longitudinal three-level models with clustering due to therapists, schools, provider etc, are supported. Random slopes at the subject level and cluster level are possible. Cluster sizes can be unbalanced, and vary by treatment. Partially nested designs are supported. Missing data can also be accounted for.

### Usage

```
study_parameters(n1, n2, n3 = 1, T_end = NULL, fixed_intercept = 0L,
                 fixed_slope = 0L, sigma_subject_intercept = NULL,
                 sigma_subject_slope = NULL, sigma_cluster_intercept = NULL,
                 sigma_cluster_slope = NULL, sigma_error = 10, cor_subject = 0L,
                 cor_cluster = 0L, cor_within = 0L, var_ratio = NULL, icc_slope = NULL,
                 icc_pre_subject = NULL, icc_pre_cluster = NULL, effect_size = 0L,
                 cohend = NULL, partially_nested = FALSE, dropout = 0L,
                 deterministic_dropout = TRUE)
```



**Arguments**

n1	Number of level 1 units, e.g. measurements per subject.
n2	Number of level 2 units per level 3 unit, e.g. subjects per cluster. Unbalanced cluster sizes are supported, see <a href="#">unequal_clusters</a> .
n3	Number of level 3 units per treatment, can be different in each treatment arm, see <a href="#">per_treatment</a> .
T_end	Time point of the last measurement. If NULL it will be set to $n1 - 1$ .
fixed_intercept	Average baseline value, assumed to be equal for both groups.
fixed_slope	Overall change per unit time, in the control group.
sigma_subject_intercept	Subject-level random intercept.
sigma_subject_slope	Subject-level random slope.
sigma_cluster_intercept	Cluster-level random intercept.
sigma_cluster_slope	Cluster-level random slope.
sigma_error	Within-subjects (residual) variation.
cor_subject	Correlation between the subject-level random intercept and slopes.
cor_cluster	Correlation between the cluster-level random intercept and slopes.
cor_within	Correlation of the level 1 residual. Currently ignored in the analytical power calculations.
var_ratio	Ratio of the random slope variance to the within-subject variance.
icc_slope	Proportion of slope variance at the cluster level.
icc_pre_subject	Amount of baseline variance at the subject level. N.B. the variance at the subject-level also included the cluster-level variance. If there's no random slopes, this would be the subject-level ICC, i.e. correlation between time points.
icc_pre_cluster	Amount of baseline variance at the cluster level.
effect_size	The treatment effect. Either a numeric indicating the mean difference (unstandardized) between the treatments at posttest, or a standardized effect using the <a href="#">cohend</a> helper function.
cohend	<i>Deprecated</i> ; now act as a shortcut to <a href="#">cohend</a> helper function. Equivalent to using <code>effect_size = cohend(cohend, standardizer = "pretest_SD", treatment = "control")</code>
partially_nested	logical; indicates if there's clustering in both arms or only in the treatment arm.
dropout	Dropout process, see <a href="#">dropout_weibull</a> or <a href="#">dropout_manual</a> . Assumed to be 0 if NULL.
deterministic_dropout	logical; if FALSE the input to dropout will be treated as random and dropout will be sampled from a multinomial distribution. N.B.: the random dropout will be sampled independently in both treatment arms.

## Details

### Comparing a combination of parameter values

It is possible to setup a grid of parameter combinations by entering the values as vectors. All unique combinations of the inputs will be returned. This is useful if you want see how different values of the parameters affect power. See also the convenience function [get\\_power\\_table](#).

### Standardized and unstandardized inputs

All parameters of the models can be specified. However, many of the raw parameter values in a multilevel/LMM do not directly affect the power of the test of the `treatment:time`-coefficient. Power will depend greatly on the relative size of the parameters, therefore, it is possible to setup your calculations using only standardized inputs, or by a combination of raw inputs and standardized inputs. For instance, if `sigma_subject_slope` and `icc_slope` is specified, the `sigma_cluster_slope` will be solved for. Only the cluster-level parameters can be solved when standardized and raw values are mixed. `sigma_error` is 10 by default. More information regarding the standardized inputs are available in the two-level and three-level vignettes.

### Difference between 0 and NA

For the variance components  $\emptyset$  and NA/NULL have different meanings. A parameter that is 0 is still kept in the model, e.g. if `icc_pre_cluster = 0` a random intercept is estimated at the cluster level, but the true value is 0. If the argument is either NULL or NA it is excluded from the model. This choice will matter when running simulations, or if Satterthwaite *dfs* are used.

The default behavior if a parameter is not specified is that `cor_subject` and `cor_cluster` is 0, and the other variance components are NULL.

### Effect size and Cohen's d

The argument `effect_size` let's you specify the average difference in change between the treatment groups. You can either pass a numeric value to define the raw difference in means at posttest, or use a standardized effect size, see [cohend](#) for more details on the standardized effects.

The argument `cohend` is kept for legacy reasons, and is equivalent to using `effect_size = cohend(cohend, standardizer)`.

### Two- or three-level models

If either `sigma_cluster_slope` or `icc_slope` and `sigma_cluster_intercept` or `icc_pre_cluster` is NULL it will be assumed a two-level design is wanted.

### Unequal cluster sizes and unbalanced allocation

It is possible to specify different cluster sizes using [unequal\\_clusters](#). Cluster sizes can vary between treatment arms by also using [per\\_treatment](#). The number of clusters per treatment can also be set by using [per\\_treatment](#). Moreover, cluster sizes can be sampled from a distribution, and treated as a random variable. See [per\\_treatment](#) and [unequal\\_clusters](#) for examples of their use.

### Missing data and dropout

Accounting for missing data in the power calculations is possible. Currently, dropout can be specified using either [dropout\\_weibull](#) or [dropout\\_manual](#). It is possible to have different dropout patterns per treatment group using [per\\_treatment](#). See their respective help pages for examples of their use.

If `deterministic_dropout = TRUE` then the proportion of dropout is treated as fixed. However, exactly which subjects dropout is randomly sampled within treatments. Thus, clusters can become slightly unbalanced, but generally power varies little over realizations.

For *random dropout*, `deterministic_dropout = FALSE`, the proportion of dropout is converted to the probability of having exactly  $i$  measurements, and the actual dropout is sampled from a multinomial distribution. In this case, the proportion of dropout varies over the realizations from the multinomial distribution, but will match the dropout proportions in expectation. The random dropout in each treatment group is sampled from independent multinomial distributions.

Generally, power based on fixed dropout is a good approximation of random dropout.

### Value

A list or data.frame of parameters values, either of class `plcp` or `plcp_multi` if multiple parameters are compared.

### See Also

[cohend](#), [get\\_power](#), [simulate.plcp](#)

### Examples

```
# Three level model with both subject- and cluster-level random slope
# Power calculation using standardized inputs
p <- study_parameters(n1 = 11,
```

```
  n2 = 5,
  n3 = 4,
  icc_pre_subject = 0.5,
  icc_pre_cluster = 0,
  var_ratio = 0.03,
  icc_slope = 0.05,
  effect_size = cohend(-0.8))
```

```
get_power(p)
```

```
# The same calculation with all parameters specified directly
```

```
p <- study_parameters(n1 = 11,
  n2 = 5,
  n3 = 4,
  T_end = 10,
  fixed_intercept = 37,
  fixed_slope = -0.65,
  sigma_subject_intercept = 2.8,
  sigma_subject_slope = 0.4726944,
  sigma_cluster_intercept = 0,
  sigma_cluster_slope = 0.1084435,
  sigma_error = 2.8,
  cor_subject = -0.5,
  cor_cluster = 0,
  effect_size = cohend(-0.8))
```

```
get_power(p)
```

```
# Standardized and unstandardized inputs
```

```
p <- study_parameters(n1 = 11,
  n2 = 5,
  n3 = 4,
```

```

        sigma_subject_intercept = 2.8,
        icc_pre_cluster = 0.07,
        sigma_subject_slope = 0.47,
        icc_slope = 0.05,
        sigma_error = 2.8,
        effect_size = cohend(-0.8))

get_power(p)

## Two-level model with subject-level random slope
p <- study_parameters(n1 = 11,
                     n2 = 40,
                     icc_pre_subject = 0.5,
                     var_ratio = 0.03,
                     effect_size = cohend(-0.8))

get_power(p)

# add missing data
p <- update(p, dropout = dropout_weibull(0.2, 1))
get_power(p)

## Comparing a combination of values
p <- study_parameters(n1 = 11,
                     n2 = c(5, 10),
                     n3 = c(2, 4),
                     icc_pre_subject = 0.5,
                     icc_pre_cluster = 0,
                     var_ratio = 0.03,
                     icc_slope = c(0, 0.05),
                     effect_size = cohend(c(-0.5, -0.8))
                     )

get_power(p)

```

---

```
summary.plcp_multi_sim
```

*Summarize simulations based on a combination of multiple parameter values*

---

## Description

Summarize simulations based on a combination of multiple parameter values

## Usage

```
## S3 method for class 'plcp_multi_sim'
summary(object, para = "time:treatment",
        type = "fixed", model = "correct", alpha = 0.05, ...)
```

**Arguments**

object	A multiple simulation object created with <code>simulate.plcp_multi</code>
para	The name of the fixed or random effect that should be summarized.
type	Specifies what type of effect para is; can be either "random" or "fixed".
model	Specifies which model should be summarized. Used when models are simulated from formulas named "correct" and "wrong".
alpha	Indicates the significance level. Default is 0.05 (two-tailed), one-tailed tests are not yet implemented.
...	Optional arguments.

**Value**

A data.frame with class `plcp_multi_sim_summary`. Each row summarizes one of the parameter combinations used in the simulation. It contains the following columns:

- parameter is the name of the coefficient
- M\_est is the mean of the estimates taken over all the simulations.
- theta is the population parameter values specified with `study_parameters`
- est\_rel\_bias is the relative bias of the estimate
- M\_se is the mean estimated standard error taken over all the simulations.
- SD\_est is the empirical standard error; i.e. the standard deviation of the distribution of the generated estimates
- se\_rel\_bias is the relative bias of the standard error.
- power is the empirical power of the Wald Z test, i.e. the proportion of simulated p-values < alpha
- power\_satt is the empirical power of the Wald *t* test using Satterthwaite's degree of freedom approximation
- satt\_NA is the proportion of Satterthwaite's approximations that failed.
- prop\_zero is the proportion of the simulated estimates that are zero; only shown for random effects

---

summary.plcp_sim	<i>Summarize the results from a simulation of a single study design-object</i>
------------------	--

---

**Description**

Summarize the results from a simulation of a single study design-object

**Usage**

```
## S3 method for class 'plcp_sim'
summary(object, alpha = 0.05, ...)
```

**Arguments**

object	A <code>simulate.plcp</code> -object
alpha	Indicates the significance level. Default is 0.05 (two-tailed), one-tailed tests are not yet implemented.
...	Currently not used

**Value**

Object with class `plcp_sim_summary`. It contains the following output:

- `parameter` is the name of the coefficient
- `M_est` is the mean of the estimates taken over all the simulations.
- `M_se` is the mean estimated standard error taken over all the simulations.
- `SD_est` is the empirical standard error; i.e. the standard deviation of the distribution of the generated estimates.
- `power` is the empirical power of the Wald Z test, i.e. the proportion of simulated p-values < alpha.
- `power_satt` is the empirical power of the Wald *t* test using Satterthwaite's degree of freedom approximation.
- `satt_NA` is the proportion of Satterthwaite's approximations that failed.
- `prop_zero` is the proportion of the simulated estimates that are zero; only shown for random effects.

---

`unequal_clusters`      *Setup unbalanced cluster sizes*

---

**Description**

Helps specifying unequal cluster sizes with [study\\_parameters](#)

**Usage**

```
unequal_clusters(..., func = NULL, trunc = 1, replace = 1)
```

**Arguments**

...	Any number of separate numeric arguments specifying each cluster's size
func	A function that generates cluster sizes, used instead of ... See <i>Details</i> .
trunc	Cutoff for values generated by <code>func</code> , $x < \text{trunc}$ are replaced, used to avoid negative or 0 values.
replace	Indicates what value to replace cluster sizes less than <code>trunc</code> with.

**Details**

If `func` is used together with a function that generates random draws, e.g. `rnorm` or `rpois`, then cluster sizes (and possibly the number of clusters), will be treated as a random variable. The expected power is then reported by averaging over multiple realizations of the random variables.

Unless `per_treatment` is used, then the same realization of random cluster sizes will be used in both groups. To use independent realizations from the same distribution for each treatment group, simply combine the `unequal_clusters` with `per_treatment`.

**Value**

An object of type `'plcp_unequal_clusters'`

**See Also**

[per\\_treatment](#)

**Examples**

```
library(dplyr)
n2 <- unequal_clusters(5, 10, 15, 40)
p <- study_parameters(n1 = 11,
                     n2 = n2,
                     n3 = 6,
                     T_end = 10,
                     icc_pre_subject = 0.5,
                     icc_pre_cluster = 0,
                     sigma_error = 1,
                     var_ratio = 0.03,
                     icc_slope = 0.05,
                     cohend = -0.8)

# verify cluster sizes
d <- simulate_data(p)
d %>%
  filter(time == 0) %>%
  group_by(treatment, cluster) %>%
  summarise(n = n())

# Poisson distributed cluster sizes, same in both groups
n2 <- unequal_clusters(func = rpois(n = 5, lambda = 5))
p <- study_parameters(n1 = 11,
                     n2 = n2,
                     T_end = 10,
                     icc_pre_subject = 0.5,
                     icc_pre_cluster = 0,
                     sigma_error = 1,
                     var_ratio = 0.03,
                     icc_slope = 0.05,
                     cohend = -0.8)

# Independent draws from same dist
```

```

n2 <- unequal_clusters(func = rpois(n = 5, lambda = 5))
p <- study_parameters(n1 = 11,
                     n2 = per_treatment(n2, n2),
                     T_end = 10,
                     icc_pre_subject = 0.5,
                     icc_pre_cluster = 0,
                     sigma_error = 1,
                     var_ratio = 0.03,
                     icc_slope = 0.05,
                     cohend = -0.8)

# Use per_treatment() to specify per treatment -----
n2 <- per_treatment(unequal_clusters(2, 2, 2, 2, 3, 4, 5),
                  unequal_clusters(10, 15))
p <- study_parameters(n1 = 11,
                     n2 = n2,
                     n3 = 3,
                     T_end = 10,
                     icc_pre_subject = 0.5,
                     icc_pre_cluster = 0,
                     var_ratio = 0.03,
                     icc_slope = 0.05,
                     cohend = -0.8)

# verify cluster sizes
d <- simulate_data(p)
d %>%
  filter(time == 0) %>%
  group_by(treatment, cluster) %>%
  summarise(n = n())

```

---

update.plcp

*Update a study\_parameters-object with new settings*


---

## Description

Update a study\_parameters-object with new settings

## Usage

```
## S3 method for class 'plcp'
update(object, ...)
```

## Arguments

object	An object created by <a href="#">study_parameters</a>
...	Any number of named arguments that should be updated



**Details**

Currently only the arguments used to construct the original object can be updated.

**Examples**

```
p <- study_parameters(n1 = 11,
                    n2 = 10,
                    n3 = 3,
                    T_end = 10,
                    icc_pre_subject = 0.5,
                    icc_pre_cluster = 0,
                    var_ratio = 0.03,
                    icc_slope = 0.05,
                    cohend = -0.8)

p <- update(p, icc_slope = 0.1)
get_ICC_slope(p)

## Not run:
# Using a "new" argument does not work (yet)
update(p, sigma_cluster_slope = 2)

## End(Not run)
```

---

[.plcp\_multi\_power      *Subset function for plcp\_multi\_power-objects*

---

**Description**

Custom subset function for plcp\_multi\_power-object to make it compatible with its print method.

**Usage**

```
## S3 method for class 'plcp_multi_power'
x[i, ...]
```

**Arguments**

x	A plcp_multi_power-object.
i	Indicates which rows to subset.
...	Ignored.

# Index

[.plcp\_multi\_power, 49

cohend, 3, 41–43

create\_lmer\_formula, 6, 36

dropout\_manual, 6, 8, 12, 41, 42

dropout\_weibull, 7, 8, 12, 24, 41, 42

get\_correlation\_matrix, 10, 20, 25, 29

get\_DEFT, 11

get\_dropout, 12

get\_ICC\_pre\_clusters, 13

get\_ICC\_pre\_subjects, 14

get\_ICC\_slope, 14

get\_monte\_carlo\_se, 15, 29

get\_power, 15, 16, 19, 28, 43

get\_power\_table, 17, 18, 42

get\_sds, 20

get\_slope\_diff, 21

get\_var\_ratio, 21

get\_VPC, 20, 22, 26

per\_treatment, 7, 8, 23, 41, 42, 47

plot.plcp, 24

plot.plcp\_ICC2, 25

plot.plcp\_power\_table, 25

plot.plcp\_sds, 26

plot.plcp\_VPC, 23, 26

powerlmm, 27

powerlmm-package (powerlmm), 27

print.plcp\_2lv1, 28

print.plcp\_3lv1, 28

print.plcp\_ICC2, 29

print.plcp\_mc\_se, 29

print.plcp\_multi, 30

print.plcp\_multi\_power, 30

print.plcp\_multi\_sim, 31

print.plcp\_multi\_sim\_summary, 31

print.plcp\_power\_2lv1, 32

print.plcp\_power\_3lv1, 32

print.plcp\_sds, 33

print.plcp\_sim, 33

print.plcp\_sim\_summary, 34

print.plcp\_VPC, 34

shiny\_powerlmm, 35

simulate.plcp, 11, 17, 33, 35, 43

simulate.plcp\_multi, 31, 45

simulate.plcp\_multi (simulate.plcp), 35

simulate\_data, 37, 39

study\_parameters, 4, 6, 8, 10–17, 19–24, 27, 28, 36, 39, 40, 46, 48

summary.plcp\_multi\_sim, 44

summary.plcp\_sim, 36, 37, 45

unequal\_clusters, 24, 41, 42, 46

update.plcp, 48