Package ‘pwr4exp’

November 11, 2025

Title Power Analysis for Research Experiments

Version 1.0.1

Description Provides tools for calculating statistical power for experiments analyzed using linear mixed models. It supports standard designs, including randomized block, split-plot, and Latin Square designs, while offering flexibility to accommodate a variety of other complex study designs.

License GPL (>= 2)

URL https://github.com/an-ethz/pwr4exp,
https://an-ethz.github.io/pwr4exp/

BugReports https://github.com/an-ethz/pwr4exp/issues

Depends R (>= 2.10), stats

Suggests agricolae, AlgDesign, crossdes, FrF2, knitr, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Imports emmeans, MASS, Matrix, methods, nlme, numDeriv

NeedsCompilation no

Author Kai Wang [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-6672-1121>),
Mutian Niu [aut, cph] (ORCID: <https://orcid.org/0000-0003-4484-4611>)

Maintainer Kai Wang <kai.wang@usys.ethz.ch>

Repository CRAN

Date/Publication 2025-11-11 11:50:02 UTC

Contents

---

corAR1 — *AR(1) Correlation Structure*

Description

Re-exports nlme::corAR1 from the **nlme** package.

Usage

```
corAR1(value, form, fixed)
```

Arguments

`value`	numeric value for the correlation parameter.
`form`	a one-sided formula of the form ~ `t`, ~ `1	g`, or ~ `t	g`, specifying a time covariate `t`, a grouping factor g, or both. When no time covariate is specified, the row order of the data within each group is assumed to represent the chronological order of measurements.
`fixed`	unused

Details

In the original nlme::corAR1 function, a covariate `t` must be an integer class. In **pwr4exp**, `t` can also be a factor class, which is then converted to an integer internally for chronological order. The class of `t` in the model formula, if present, is not converted. For example, a time covariate can be fitted as a factor in the model formula, whereas it is converted to an integer in the correlation formula.

See Also

corClasses See nlme::corAR1 for original documentation.

`corARMA` *ARMA(p,q) Correlation Structure*

Description

Re-exports nlme::corARMA from the **nlme** package.

Usage

```
corARMA(value, form, p, q, fixed)
```

Arguments

`value`	numeric vector of parameter values
`form`	a one-sided formula of the form ~ `t`, ~ `1	g`, or ~ `t	g`, specifying a time covariate `t`, a grouping factor `g`, or both. When no time covariate is specified, the row order of the data within each group is assumed to represent the chronological order of measurements.
`p`	non-negative integer specifying the autoregressive order
`q`	non-negative integer specifying the moving average order
`fixed`	unused

Details

In the original nlme::corARMA function, a covariate `t` must be an integer class. In **pwr4exp**, `t` can also be a factor class, which is then converted to an integer internally for chronological order. The class of `t` in the model formula, if present, is not converted. For example, a time covariate can be fitted as a factor in the model formula, whereas it is converted to an integer in the correlation formula.

See Also

corClasses See nlme::corARMA for original documentation.

`corCAR1` *Continuous AR(1) Correlation Structure*

Description

Re-exports nlme::corCAR1 from the **nlme** package.

Usage

```
corCAR1(value, form, fixed)
```

Arguments

`value`	numeric value for the correlation parameter
`form`	a one-sided formula of the form `~ t`, `~ 1 | g`, or `~ t | g`, specifying a time covariate `t`, a grouping factor `g`, or both.
`fixed`	unused

See Also

corClasses See nlme::corCAR1 for original documentation.

`corClasses` *Correlation Structure Classes*

Description

Standard classes of correlation structures (`corStruct`) available from the **nlme** package and re-exported by **pwr4exp** for convenience when specifying correlation structures in `mkdesign`.

All arguments are identical to the corresponding **nlme** functions. For more details on the original implementations, see nlme::corClasses.

Note: In the original nlme::corAR1, nlme::corARMA, and nlme::corSymm functions, the covariate `t` in the correlation formula `~ t` or `~ t | g` must be an integer class. In **pwr4exp**, the covariate can also be a factor class, which is then converted to an integer internally for sorting purposes. The class of the covariate variable in the model formula, if present, will not be converted. For example, a time covariate can be fitted as a factor in the model formula, whereas it is converted to an integer in the correlation formula temporarily for matrix sorting.

Details

Available standard classes:

`corAR1` autoregressive process of order 1.

`corARMA` autoregressive moving average process, with arbitrary orders for the autoregressive and moving average components.

`corCAR1` continuous AR(1)

`corCompSymm` compound symmetry structure corresponding to a constant correlation.

`corExp` exponential spatial correlation.

`corGaus` Gaussian spatial correlation.

`corLin` linear spatial correlation.

`corRatio` Rational quadratics spatial correlation.

`corSpher` spherical spatial correlation.

`corSymm` general correlation matrix, with no additional structure.

References

Pinheiro, J.C., and Bates, D.M. (2000) "Mixed-Effects Models in S and S-PLUS", Springer.

`corCompSymm` *Compound Symmetry Correlation Structure*

Description

Re-exports nlme::corCompSymm from the **nlme** package.

Usage

```
corCompSymm(value, form, fixed)
```

Arguments

`value`	numeric vector of parameter values
`form`	a one-sided formula of the form `~ 1	g` specifying a grouping factor.
`fixed`	unused

See Also

corClasses See nlme::corCompSymm for original documentation.

corExp *Exponential Correlation Structure*

Description

Re-exports nlme::corExp from the **nlme** package.

Usage

```
corExp(value, form, nugget, metric, fixed)
```

Arguments

`value`	numeric value(s) for the parameter(s)
`form`	a one-sided formula of the form `~ S1 + ... + Sp`, or `~ S1 + ... + Sp | g` spatial covariates S1 through Sp and, optionally, a grouping factor g.
`nugget`	logical; if `TRUE` a nugget effect is added
`metric`	character string specifying the distance metric
`fixed`	unused

See Also

corClasses See nlme::corExp for original documentation.

corGaus *Gaussian Correlation Structure*

Description

Re-exports nlme::corGaus from the **nlme** package.

Usage

```
corGaus(value, form, nugget, metric, fixed)
```

Arguments

`value`	numeric value(s) for the parameter(s)
`form`	a one-sided formula of the form `~ S1 + ... + Sp`, or `~ S1 + ... + Sp | g` spatial covariates S1 through Sp and, optionally, a grouping factor g.
`nugget`	logical; if `TRUE` a nugget effect is added
`metric`	character string specifying the distance metric
`fixed`	unused

See Also

corClasses See nlme::corGaus for original documentation.

`corLin` *Linear Correlation Structure*

Description

Re-exports nlme::corLin from the **nlme** package.

Usage

```
corLin(value, form, nugget, metric, fixed)
```

Arguments

`value`	numeric value(s) for the parameter(s)
`form`	a one-sided formula of the form `~ S1 + ... + Sp`, or `~ S1 + ... + Sp | g` spatial covariates S1 through Sp and, optionally, a grouping factor g.
`nugget`	logical; if `TRUE` a nugget effect is added
`metric`	character string specifying the distance metric
`fixed`	unused

See Also

corClasses See nlme::corLin for original documentation.

`corRatio` *Rational Quadratics Correlation Structure*

Description

Re-exports nlme::corRatio from the **nlme** package.

Usage

```
corRatio(value, form, nugget, metric, fixed)
```

Arguments

`value`	numeric value(s) for the parameter(s)
`form`	a one-sided formula of the form `~ S1 + ... + Sp`, or `~ S1 + ... + Sp | g` spatial covariates S1 through Sp and, optionally, a grouping factor g.
`nugget`	logical; if `TRUE` a nugget effect is added
`metric`	character string specifying the distance metric
`fixed`	unused

See Also

corClasses See nlme::corRatio for original documentation.

`corSpher` *Spherical Correlation Structure*

Description

Re-exports nlme::corSpher from the **nlme** package.

Usage

```
corSpher(value, form, nugget, metric, fixed)
```

Arguments

`value`	numeric value(s) for the parameter(s)
`form`	a one-sided formula of the form ~ `S1 + ... + Sp`, or ~ `S1 + ... + Sp	g` spatial covariates S1 through Sp and, optionally, a grouping factor g.
`nugget`	logical; if `TRUE` a nugget effect is added
`metric`	character string specifying the distance metric
`fixed`	unused

See Also

corClasses See nlme::corSpher for original documentation.

`corSymm` *General Correlation Structure*

Description

Re-exports nlme::corSymm from the **nlme** package.

Usage

```
corSymm(value, form, fixed)
```

Arguments

`value`	numeric vector of correlation parameter values
`form`	a one-sided formula of the form ~ `t`, ~ `1	g`, or ~ `t	g`, specifying a covariate `t`, a grouping factor g, or both. A covariate indicates the order of the data rows in the residual error matrix of each group.
`fixed`	unused

See Also

corClasses See nlme::corSymm for original documentation.

`designCRD` *Creation of Standard Experimental Designs*

Description

These functions facilitate the creation of standard experimental designs commonly used in agricultural studies for power analysis. Unlike mkdesign which requires a pre-existing data frame, these functions allow users to directly specify key design characteristics to generate experimental layouts. Quantitative parameters describing fixed and random effects remain consistent with those in mkdesign.

Usage

```
designCRD(
  treatments,
  label,
  replicates,
  formula,
  beta = NULL,
  means = NULL,
  sigma2,
  template = FALSE,
  REML = TRUE
)

designRCBD(
  treatments,
  label,
  blocks,
  formula,
  beta = NULL,
  means = NULL,
  vcomp,
  sigma2,
  template = FALSE,
  REML = TRUE
)

designLSD(
  treatments,
  label,
  squares = 1,
  reuse = c("row", "col", "none"),
  formula,
  beta = NULL,
  means = NULL,
  vcomp,
  sigma2,
  template = FALSE,
  REML = TRUE
)

designCOD(
  treatments,
  label,
  squares = 1,
  formula,
  beta = NULL,
  means = NULL,
  vcomp,
  sigma2,
  template = FALSE,
  REML = TRUE
)

designSPD(
  trt.main,
  trt.sub,
  label,
  replicates,
  formula,
  beta = NULL,
  means = NULL,
  vcomp,
  sigma2,
  template = FALSE,
  REML = TRUE
)
```

Arguments

`treatments`	An integer vector where each element represents the number of levels of the corresponding treatment factor. A single integer (e.g., `treatments = n`) specifies one treatment factor with `n` levels. When multiple factors are provided, they are arranged in a factorial treatment factor design. For example, `treatments = c(2, 3)` creates a 2x3 factorial design with the first factor having 2 levels and the second factor having 3 levels.
`label`	Optional. A list of character vectors, each corresponding to a treatment factor. The name of each vector specifies the factor's name, and its elements provide the labels for that factor's levels. If no labels are provided, default labels will be used. For a single treatment factor, the default is `list(trt = c("1", "2",`

	`...))`, and for two treatment factors, the default is `list(facA = c("1", "2", ...), facB = c("1", "2", ...))`. For split-plot designs, the defaults are similar but include the ".main" and ".sub" suffixes for main plot and subplot factors. For example: `list(trt.main = c("1", "2", ...), trt.sub = c("1", "2", ...))` and `list(facA.main = c("1", "2", ...), facB.main = c("1", "2", ...), facA.sub = c("1", "2", ...), facB.sub = c("1", "2", ...))`. Label sets should be arranged so that the main plot factors come first, followed by the subplot factors.
`replicates`	The number of experimental units per treatment in a completely randomized design or the number of experimental units (main plots) per treatment of main plot factors.
`formula`	A right-hand-side formula specifying the model for testing treatment effects, with terms on the right of ~ , following lme4::lmer syntax for random effects. If not specified, a default formula with main effects and all interactions is used internally.
`beta`	One of the optional inputs for fixed effects. A vector of model coefficients where factor variable coefficients correspond to dummy variables created using treatment contrast (stats::contr.treatment).
`means`	One of the optional inputs for fixed effects. A vector of marginal or conditioned means (if factors have interactions). Regression coefficients are required for numerical variables. Either `beta` or `means` must be provided, and their values must strictly follow a specific order. A template can be created to indicate the required input values and their order. See mkdesign for more information.
`sigma2`	error variance.
`template`	Default is `FALSE`. If `TRUE`, a template for `beta`, `means`, and `vcomp` is generated to indicate the required input order.
`REML`	Specifies whether to use REML or ML information matrix. Default is `TRUE` (REML).
`blocks`	The number of blocks.
`vcomp`	A vector of variance-covariance components for random effects, if present. The values must follow a strict order. See mkdesign.
`squares`	The number of replicated squares. By default, 1, i.e., no replicated squares.
`reuse`	A character string specifying how to replicate squares when there are multiple squares. Options are: "row" for reusing row blocks, "col" for reusing column blocks, or "none" for reusing neither row nor column blocks to replicate a single square.
`trt.main`	An integer-valued vector specifying the treatment structure at main plot level for a split plot design, similar to `treatments`.
`trt.sub`	An integer-valued vector specifying the treatment structure at sub plot level for a split plot design, similar to `treatments`.

Details

Each function creates a standard design as described below:

designCRD Completely Randomized Design. By default, the model formula is ~ trt for one factor and ~ facA*facB for two factors, unless explicitly specified. If the label argument is provided, the formula is automatically updated with the specified treatment factor names.

designRCBD Randomized Complete Block Design. Experimental units are grouped into blocks, with each treatment appearing **exactly** once per block (i.e., no replicates per treatment within a block). The default model formula is ~ trt + (1|block) for one factor and ~ facA*facB + (1|block) for two factors. If label is provided, the fixed effect parts of the formula are automatically updated with the specified names. The block factor is named "block" and not changeable.

designLSD Latin Square Design. The default formula is ~ trt + (1|row) + (1|col) for one factor and ~ facA*facB + (1|row) + (1|col) for two factors. If label is provided, the fixed effect parts of the formula are automatically updated with the specified names. The names of row ("row") and column ("col") block factors are not changeable.

designCOD Crossover Design, which is a special case of LSD with time periods and individuals as blocks. Period blocks are reused when replicating squares. The default formula is ~ trt + (1|subject) + (1|period) for one factor and ~ facA*facB + (1|subject) + (1|period) for two factors. If label is provided, the fixed effect parts of the formula are automatically updated with the specified names. Note that "subject" and "period" are the names for the two blocking factors and cannot be changed.

designSPD Split Plot Design. The default formula includes the main effects of all treatment factors at both the main and sub-plot levels, their interactions, and the random effects of main plots: ~ . + (1|mainplot). If label is provided, the fixed effect parts of the formula are automatically updated with the specified names. The experimental unit at the main plot level (i.e., the block factor at the subplot level) is always named as "mainplot".

Value

A list object containing all essential components for power calculation. This includes:

- Structural components (deStruct): including the data frame, design matrices for fixed and random effects, variance-covariance matrices for random effects and residuals, etc.
- Internally calculated higher-level parameters (deParam), including variance-covariance matrix of beta coefficients (vcov_beta), variance-covariance matrix of variance parameters (vcov_varpar), gradient matrices (Jac_list), etc.

See Also

mkdesign, pwr.anova, pwr.contrast

Examples

```
# Evaluate the power of a CRD with one treatment factor
## Create a design object
crd <- designCRD(
  treatments = 4, # 4 levels of one treatment factor
  replicates = 12, # 12 units per level, 48 units totally
  means = c(30, 28, 33, 35), # means of the 4 levels
  sigma2 = 10 # error variance
)
```

```
## power of omnibus test
pwr.anova(crd)

## power of contrast
pwr.contrast(crd, which = "trt", contrast = "pairwise") # pairwise comparisons
pwr.contrast(crd, which = "trt", contrast = "poly") # polynomial contrasts

# More examples are available in `vignette("pwr4exp")`
# and on https://an-ethz.github.io/pwr4exp/
```

`df.cod` *Create a data frame for Crossover design*

Description

Create a data frame for Crossover design

Usage

```
df.cod(treatments, label, squares)
```

Arguments

`treatments`	An integer vector where each element represents the number of levels of the corresponding treatment factor. A single integer (e.g., `treatments = n`) specifies one treatment factor with `n` levels. When multiple factors are provided, they are arranged in a factorial treatment factor design. For example, `treatments = c(2, 3)` creates a 2x3 factorial design with the first factor having 2 levels and the second factor having 3 levels.
`label`	Optional. A list of character vectors, each corresponding to a treatment factor. The name of each vector specifies the factor's name, and its elements provide the labels for that factor's levels. If no labels are provided, default labels will be used. For a single treatment factor, the default is `list(trt = c("1", "2", ...))`, and for two treatment factors, the default is `list(facA = c("1", "2", ...), facB = c("1", "2", ...))`. For split-plot designs, the defaults are similar but include the ".main" and ".sub" suffixes for main plot and subplot factors. For example: `list(trt.main = c("1", "2", ...), trt.sub = c("1", "2", ...))` and `list(facA.main = c("1", "2", ...), facB.main = c("1", "2", ...), facA.sub = c("1", "2", ...), facB.sub = c("1", "2", ...))`. Label sets should be arranged so that the main plot factors come first, followed by the subplot factors.
`squares`	The number of replicated squares. By default, 1, i.e., no replicated squares.

Value

a data.frame representing the data structure of the design

df.crd *Create a data frame of completely randomized design*

Description

Create a data frame of completely randomized design

Usage

```
df.crd(treatments, label, replicates)
```

Arguments

`treatments`	An integer vector where each element represents the number of levels of the corresponding treatment factor. A single integer (e.g., `treatments = n`) specifies one treatment factor with `n` levels. When multiple factors are provided, they are arranged in a factorial treatment factor design. For example, `treatments = c(2, 3)` creates a 2x3 factorial design with the first factor having 2 levels and the second factor having 3 levels.
`label`	Optional. A list of character vectors, each corresponding to a treatment factor. The name of each vector specifies the factor's name, and its elements provide the labels for that factor's levels. If no labels are provided, default labels will be used. For a single treatment factor, the default is `list(trt = c("1", "2", ...))`, and for two treatment factors, the default is `list(facA = c("1", "2", ...), facB = c("1", "2", ...))`. For split-plot designs, the defaults are similar but include the ".main" and ".sub" suffixes for main plot and subplot factors. For example: `list(trt.main = c("1", "2", ...), trt.sub = c("1", "2", ...))` `list(facA.main = c("1", "2", ...), facB.main = c("1", "2", ...), facA.sub = c("1", "2", ...), facB.sub = c("1", "2", ...))` Label sets should be arranged so that the main plot factors come first, followed by the subplot factors.
`replicates`	The number of experimental units per treatment.

Value

a data.frame representing the data structure of the design

df.lsd *Create a data frame for Latin square design*

Description

Create a data frame for Latin square design

Usage

```
df.lsd(treatments, label, squares = 1, reuse = c("row", "col", "none"))
```

Arguments

`treatments`	An integer vector where each element represents the number of levels of the corresponding treatment factor. A single integer (e.g., `treatments = n`) specifies one treatment factor with `n` levels. When multiple factors are provided, they are arranged in a factorial treatment factor design. For example, `treatments = c(2, 3)` creates a 2x3 factorial design with the first factor having 2 levels and the second factor having 3 levels.
`label`	Optional. A list of character vectors, each corresponding to a treatment factor. The name of each vector specifies the factor's name, and its elements provide the labels for that factor's levels. If no labels are provided, default labels will be used. For a single treatment factor, the default is `list(trt = c("1", "2", ...))`, and for two treatment factors, the default is `list(facA = c("1", "2", ...), facB = c("1", "2", ...))`. For split-plot designs, the defaults are similar but include the ".main" and ".sub" suffixes for main plot and subplot factors. For example: `list(trt.main = c("1", "2", ...), trt.sub = c("1", "2", ...))` and `list(facA.main = c("1", "2", ...), facB.main = c("1", "2", ...), facA.sub = c("1", "2", ...), facB.sub = c("1", "2", ...))`. Label sets should be arranged so that the main plot factors come first, followed by the subplot factors.
`squares`	the number of replicated squares
`reuse`	A character string specifying how to replicate squares when there are multiple squares. Options are: "row" for reusing row blocks, "col" for reusing column blocks, or "none" for reusing neither row nor column blocks to replicate a single square.

Value

a data.frame representing the data structure of the design

`df.rcbd` *Create a data frame of randomized complete block design*

Description

Create a data frame of randomized complete block design

Usage

```
df.rcbd(treatments, label, blocks)
```

Arguments

`treatments`	An integer vector where each element represents the number of levels of the corresponding treatment factor. A single integer (e.g., `treatments = n`) specifies one treatment factor with `n` levels. When multiple factors are provided, they are arranged in a factorial treatment factor design. For example, `treatments = c(2, 3)` creates a 2x3 factorial design with the first factor having 2 levels and the second factor having 3 levels.
`label`	Optional. A list of character vectors, each corresponding to a treatment factor. The name of each vector specifies the factor's name, and its elements provide the labels for that factor's levels. If no labels are provided, default labels will be used. For a single treatment factor, the default is `list(trt = c("1", "2", ...))`, and for two treatment factors, the default is `list(facA = c("1", "2", ...), facB = c("1", "2", ...))`. For split-plot designs, the defaults are similar but include the ".main" and ".sub" suffixes for main plot and subplot factors. For example: `list(trt.main = c("1", "2", ...), trt.sub = c("1", "2", ...))` and `list(facA.main = c("1", "2", ...), facB.main = c("1", "2", ...), facA.sub = c("1", "2", ...), facB.sub = c("1", "2", ...))`. Label sets should be arranged so that the main plot factors come first, followed by the subplot factors.
`blocks`	the number of blocks

Value

a data.frame representing the data structure of the design

df.spd *Create data frame for split-plot design*

Description

Create data frame for split-plot design

Usage

```
df.spd(trt.main, trt.sub, label, replicates)
```

Arguments

`trt.main`	an integer-valued vector specifying the treatment structure at main plot level, similar to `df.crd`.
`trt.sub`	an integer-valued vector specifying the treatment structure at sub plot level, similar to `trt.main`.
`label`	Optional. A list of character vectors, each corresponding to a treatment factor. The name of each vector specifies the factor's name, and its elements provide the labels for that factor's levels. If no labels are provided, default labels will be used. For a single treatment factor, the default is `list(trt = c("1", "2",`

...)), and for two treatment factors, the default is `list(facA = c("1", "2", ...), facB = c("1", "2", ...))`. For split-plot designs, the defaults are similar but include the ".main" and ".sub" suffixes for main plot and subplot factors. For example: `list(trt.main = c("1", "2", ...), trt.sub = c("1", "2", ...))` and `list(facA.main = c("1", "2", ...), facB.main = c("1", "2", ...), facA.sub = c("1", "2", ...), facB.sub = c("1", "2", ...))`. Label sets should be arranged so that the main plot factors come first, followed by the subplot factors.

`replicates`	the number of experimental units (main plots) per treatment of main plot factors.

Value

a data.frame representing the data structure of the design

`milk` *An exemplary dataset of a 4x4 crossover design with 2 squares*

Description

Milk yield records from 8 cows over 4 different periods in a 4x4 crossover design. The design includes 2 Latin squares, each consisting of 4 cows and 4 periods.

Usage

```
milk
```

Format

A data frame with 32 rows and 4 variables:

Cow Factor: Cow index (8 levels)

Period Factor: Period index (4 levels)

Treatment Factor: Treatment index (4 levels)

MilkYield Numeric: milk yield recordings (in kg)

Source

Simulated data for package demonstration purposes.

mkdesign *Create a Design Object for Power Calculation*

Description

Generate a design object for power analysis by specifying a model formula and data frame. This object is not a true experimental design as created by design generation procedures, where randomization and unit allocation are performed. Instead, it serves as an object containing all necessary information for power analysis, including design matrices, assumed values of model effects, and other internally calculated parameters.

Usage

```
mkdesign(
  formula,
  data,
  beta = NULL,
  means = NULL,
  vcomp = NULL,
  sigma2 = NULL,
  correlation = NULL,
  template = FALSE,
  REML = TRUE
)
```

Arguments

`formula`	A right-hand-side formula specifying the model for testing treatment effects, with terms on the right of ~ , following lme4::lmer syntax for random effects.
`data`	A data frame with all independent variables specified in the model, matching the design's structure.
`beta`	One of the optional inputs for fixed effects. A vector of model coefficients where factor variable coefficients correspond to dummy variables created using treatment contrast (stats::contr.treatment).
`means`	One of the optional inputs for fixed effects. A vector of marginal or conditioned means (if factors have interactions). Regression coefficients are required for numerical variables. Either `beta` or `means` must be provided, and their values must strictly follow a specific order. A template can be created to indicate the required input values and their order. See "Details" for more information.
`vcomp`	A vector of variance-covariance components for random effects, if present. The values must follow a strict order. See "Details".
`sigma2`	error variance.
`correlation`	Specifies residual (R-side) correlation structures using nlme::corClasses functions. See "Details" for more information.

`template`	Default is `FALSE`. If `TRUE` or when only the formula and data are provided, a template for `beta`, `means`, and `vcomp` is generated to indicate the required input order.
`REML`	Specifies whether to use REML or ML information matrix. Default is `TRUE`.

Details

- **data**: A long-format data frame is required, as typically used in R for fitting linear models. This data frame can be created manually or with the help of design creation packages such as **agricolae**, **AlgDesign**, **crossdes**, or **FrF2**. It should include all independent variables specified in the model (e.g., treatments, blocks, subjects). All the irrelevant variables not specified in the model are ignored.
- **template**: Templates are automatically generated when only the formula and data are supplied, or explicitly if `template = TRUE`. Templates serve as guides for specifying inputs:
  - **Template for** `beta`: Represents the sequence of model coefficients.
  - **Template for** `means`: Specifies the order of means (for categorical variables) and/or regression coefficients (for continuous variables), depending on the scenario:
    - *Categorical variables without interactions*: Requires marginal means for each level of the categorical variable(s).
    - *Interactions among categorical variables*: Requires conditional (cell) means for all level combinations.
    - *Numerical variables without interactions*: Requires regression coefficients. The intercept must also be included if there are no categorical variables in the model.
    - *Interactions among numerical variables*: Requires regression coefficients for both main effects and interaction terms. The intercept must also be included if there are no categorical variables in the model.
    - *Categorical-by-numerical interactions*: Requires regression coefficients for the numerical variable at each level of the categorical variable, as well as marginal means for the levels of the categorical variable.

    Note: For models containing only numerical variables, the inputs for `means` and `beta` are identical. See the "Examples" for illustrative scenarios.
  - **Template for** `vcomp`: Represents a variance-covariance matrix, where integers indicate the order of variance components in the input vector.
- **correlation**: Various residual correlation structures can be specified following the instructions from corClasses constructors.

  Note: In the original `corAR1`, `corARMA`, and `corSymm` functions, the covariate `t` in the correlation formula `~ t` or `~ t | g` must be an integer class. In **pwr4exp**, the covariate can also be a factor class, which is then converted to an integer internally for sorting purposes. The class of the covariate variable in the model formula, if present, will not be converted. For example, a time covariate can be fitted as a factor in the model formula, whereas it is converted to an integer in the correlation formula temporarily for matrix sorting.

Value

A list object containing all essential components for power calculation. This includes:

- Structural components (deStruct): including design matrices for fixed and random effects, variance-covariance matrices for random effects and residuals, etc.
- Internally calculated higher-level parameters (deParam), including variance-covariance matrix of beta coefficients (vcov_beta), variance-covariance matrix of variance parameters (vcov_varpar), gradient matrices (Jac_list), etc.

Examples

```
# Using templates for specifying "means"

# Create an example data frame with four categorical variables (factors)
# and two numerical variables
df1 <- expand.grid(
  fA = factor(1:2),
  fB = factor(1:2),
  fC = factor(1:3),
  fD = factor(1:3),
  subject = factor(1:10)
)
df1$x <- rnorm(nrow(df1))  # Numerical variable x
df1$z <- rnorm(nrow(df1))  # Numerical variable z

## Categorical variables without interactions
# Means of each level of fA and fB are required in sequence.
mkdesign(~ fA + fB, df1)$fixeff$means

## Interactions among categorical variables
# Cell means for all combinations of levels of fA and fB are required.
mkdesign(~ fA * fB, df1)$fixeff$means

## Numerical variables without and with interactions, identical to beta.
# Without interactions: Regression coefficients are required
mkdesign(~ x + z, df1)$fixeff$means

# With interactions: Coefficients for main effects and interaction terms are required.
mkdesign(~ x * z, df1)$fixeff$means

## Categorical-by-numerical interactions
# Marginal means for each level of fA, and regression coefficients for x
# at each level of fA are required.
mkdesign(~ fA * x, df1)$fixeff$means

## Three factors with interactions:
# Cell means for all 12 combinations of the levels of fA, fB, and fC are required.
mkdesign(~ fA * fB * fC, df1)

# A design that mixes the above-mentioned scenarios:
# - Interactions among three categorical variables (fA, fB, fC)
# - A categorical-by-numerical interaction (fD * x)
# - Main effects for another numerical variable (z)
# The required inputs are:
# - Cell means for all combinations of levels of fA, fB, and fC
# - Means for each level of fD
# - Regression coefficients for x at each level of fD
# - Regression coefficients for z
mkdesign(~ fA * fB * fC + fD * x + z, df1)$fixeff$means

# Using templates for specifying "vcomp"

# Assume df1 represents an RCBD with "subject" as a random blocking factor.
## Variance of the random effect "subject" (intercept) is required.
mkdesign(~ fA * fB * fC * fD + (1 | subject), df1)$varcov

# Demonstration of templates for more complex random effects
## Note: This example is a demo and statistically incorrect for this data
## (no replicates under subject*fA). It only illustrates variance-covariance templates.
## Inputs required:
## - Variance of the random intercept (1st)
## - Covariance between the intercept and "fA2" (2nd)
## - Variance of "fA2" (3rd)
mkdesign(~ fA * fB * fC * fD + (1 + fA | subject), df1)$varcov

# Power analysis for repeated measures

## Create a data frame for a CRD with repeated measures
n_subject <- 6
n_trt <- 3
n_hour <- 8
trt <- c("CON", "TRT1", "TRT2")
df2 <- data.frame(
  subject = as.factor(rep(seq_len(n_trt * n_subject), each = n_hour)), # Subject as a factor
  hour = as.factor(rep(seq_len(n_hour), n_subject * n_trt)),          # Hour as a factor
  trt = rep(trt, each = n_subject * n_hour)                         # Treatment as a factor
)

## Templates
temp <- mkdesign(formula = ~ trt * hour, data = df2)
temp$fixeff$means  # Fixed effects means template

## Create a design object
# Assume repeated measures within a subject follow an AR1 process with a correlation of 0.6
design <- mkdesign(
  formula = ~ trt * hour,
  data = df2,
  means = c(1, 2.50, 3.50,
            1, 3.50, 4.54,
            1, 3.98, 5.80,
            1, 4.03, 5.40,
            1, 3.68, 5.49,
            1, 3.35, 4.71,
            1, 3.02, 4.08,
            1, 2.94, 3.78),
  sigma2 = 2,
  correlation = corAR1(value = 0.6, form = ~ hour | subject)
)
```

```
pwr.anova(design)  # Perform power analysis

## When time is treated as a numeric variable
# Means of treatments and regression coefficients for hour at each treatment level are required
df2$hour <- as.integer(df2$hour)
mkdesign(formula = ~ trt * hour, data = df2)$fixeff$means

## Polynomial terms of time in the model
mkdesign(formula = ~ trt + hour + I(hour^2) + trt:hour + trt:I(hour^2), data = df2)$fixeff$means
```

---

pwr.anova *Power of omnibus tests*

Description

Calculates the statistical power for testing the overall effects of treatment factors and their interactions, i.e., power of F-test.

Usage

```
pwr.anova(object, sig.level = 0.05, type = c("III", "II", "I", "3", "2", "1"))
```

Arguments

`object`	a design object created in pwr4exp
`sig.level`	significance level, default 0.05
`type`	the type of ANOVA table requested, default Type III

Value

a data frame with numerator degrees of freedom (`NumDF`), denominator degrees of freedom (`DenDF`), type I error rate (`sig.level`), and `power`.

See Also

mkdesign, designCRD, designRCBD, designLSD, designCOD, designSPD, pwr.summary and pwr.contrast

Examples

```
# generate an RCBD
rcbd <- designRCBD(
  treatments = c(2, 2),
  label = list(facA = c("1", "2"), facB = c("1", "2")),
  blocks = 12,
  formula = ~ facA*facB + (1|block),
  means = c(32, 35, 30, 37),
  vcomp = 4,
  sigma2 = 6
)
# power of omnibus test
pwr.anova(rcbd)
```

pwr.contrast *Power of contrasts*

Description

Computes the statistical power of t-tests for comparisons among means.

Usage

```
pwr.contrast(
  object,
  which,
  by = NULL,
  contrast = c("pairwise", "poly", "trt.vs.ctrl"),
  sig.level = 0.05,
  p.adj = FALSE,
  alternative = c("two.sided", "one.sided"),
  strict = TRUE
)
```

Arguments

`object`	a design object created in pwr4exp
`which`	the factor of interest. Multiple factors can be combined using : or `*`, e.g., `"facA*facB"`, which represents a single factor that combines the levels of both factors.
`by`	the variable to condition on
`contrast`	A character string specifying the contrast method, one of "pairwise", "poly", or "trt.vs.ctrl". Alternatively, a numeric vector defining a single contrast or a (named) list of vectors specifying multiple custom contrasts. If a list is provided, each element must be a vector whose length matches the number of levels of the factor in each `by` group. In multi-factor scenarios, factor levels are combined and treated as a single factor.
`sig.level`	significance level, default 0.05
`p.adj`	whether the sig.level should be adjusted using the Bonferroni method, default FALSE
`alternative`	one- or two-sided test. Can be abbreviated.
`strict`	use strict interpretation in two-sided case

Value

For each by condition, returns a data frame containing the contrast value (`effect`), degrees of freedom (`df`), type I error rate (`sig.level`), `power`, and the test direction (by `alternative`). When multiple by conditions are present, the results are returned as a list.

Examples

```
rcbd <- designRCBD(
  treatments = c(2, 2),
  label = list(facA = c("1", "2"), facB = c("1", "2")),
  blocks = 12,
  formula = ~ facA*facB + (1|block),
  means = c(32, 35, 30, 37),
  vcomp = 4,
  sigma2 = 6
)

# If contrast is not specified, pairwise comparisons are conducted
pwr.contrast(rcbd, which = "facA") # Marginal effect of facA
pwr.contrast(rcbd, which = "facA", by = "facB") # Conditional effect of facA within levels of facB

# Custom contrast vector, identical to pairwise comparison
pwr.contrast(rcbd, which = "facA", contrast = c(1, -1))
pwr.contrast(rcbd, which = "facA", by = "facB", contrast = c(1, -1))

# A single factor combining two factors
pwr.contrast(
  rcbd,
  which = "facA*facB",
  contrast = list(
    A1B1vs.A2B1 = c(1, -1, 0, 0),
    A1B1vs.A2B2 = c(1, 0, 0, -1)
  )
)
```

`pwr.summary` *Power for model coefficients*

Description

Computes the statistical power for testing (t-test) model coefficients.

Usage

```
pwr.summary(object, sig.level = 0.05)
```

Arguments

`object`	design object
`sig.level`	significance level, default 0.05

Value

a data frame containing model coefficients, degrees of freedom (`df`), type I error rate (`sig.level`), power, and the test direction (`alternative`).

Examples

```
rcbd <- designRCBD(
  treatments = c(2, 2),
  label = list(facA = c("1", "2"), facB = c("1", "2")),
  blocks = 12,
  formula = ~ facA*facB + (1|block),
  means = c(32, 35, 30, 37),
  vcomp = 4,
  sigma2 = 6
)
pwr.summary(rcbd)
```

Index