Package ‘simr’

April 30, 2018

Type Package
Title Power Analysis for Generalised Linear Mixed Models by Simulation
Description Calculate power for generalised linear mixed models, using
simulation. Designed to work with models fit using the 'lme4' package.
Version 1.0.4
URL https://github.com/pitakakariki/simr
BugReports https://github.com/pitakakariki/simr/issues
License GPL (>= 2)
Depends lme4 (>= 1.1-16)
Imports binom, iterators, pbkrtest, plotrix, RLRsim, stringr, stats, methods, utils, graphics, grDevices, car
Suggests testthat, knitr, rmarkdown, lmerTest (>= 3.0-0)
LazyData true
RoxygenNote 6.0.1
VignetteBuilder knitr
NeedsCompilation no
Author Green Peter [aut, cre] (<https://orcid.org/0000-0002-0238-9852>),
MacLeod Catriona [aut],
Alday Phillip [ctb]
Maintainer Green Peter <simr.peter@gmail.com>
Repository CRAN
Date/Publication 2018-04-30 10:37:21 UTC

R topics documented:
simr-package ................................................................. 2
doFit ................................................................. 2
doSim ................................................................. 3
simr-package

simr: Simulation-based power calculations for mixed models.

Description

simr is a package that makes it easy to run simulation-based power analyses with lme4.

doFit

Fit model to a new response.

Description

This is normally an internal function, but it can be overloaded to extend simr to other packages.

Usage

doFit(y, fit, subset, ...)

Arguments

- **y**: new values for the response variable (vector or matrix depending on the model).
- **fit**: a previously fitted model object.
- **subset**: boolean vector specifying how much of the data to use. If missing, the model is fit to all the data. This argument needs to be implemented for powerCurve to work.
- **...**: additional options.

Value

a fitted model object.
**doSim**  
*Generate simulated response variables.*

**Description**  
This is normally an internal function, but it can be overloaded to extend `simr` to other packages.

**Usage**  
```r  
doSim(object, ...)  
```

**Arguments**

- `object`: an object to simulate from, usually a fitted model.
- `...`: additional options.

**Value**

A vector containing simulated response values (or, for models with a multivariate response such as binomial glm’s, a matrix of simulated response values). Suitable as input for `doFit`.

**doTest**  
*Apply a hypothesis test to a fitted model.*

**Description**  
This is normally an internal function, but it can be overloaded to extend `simr` to other packages.

**Usage**  
```r  
doTest(object, test, ...)  
```

**Arguments**

- `object`: an object to apply a statistical test to, usually a fitted model.
- `test`: a test function, see `tests`.
- `...`: additional options.

**Value**

A p-value with attributes describing the test.
extend

Extend a longitudinal model.

Description

This method increases the sample size for a model.

Usage

extend(object, along, within, n, values)

Arguments

object
  a fitted model object to extend.

along
  the name of an explanatory variable. This variable will have its number of levels extended.

within
  names of grouping variables, separated by "+" or ",". Each combination of groups will be extended to n rows.

n
  number of levels: the levels of the explanatory variable will be replaced by 1, 2, 3, \ldots, n for a continuous variable or a, b, c, \ldots, n for a factor.

values
  alternatively, you can specify a new set of levels for the explanatory variable.

Details

extend takes "slices" through the data for each unique value of the extended variable. An extended dataset is built from n slices, with slices duplicated if necessary.

Value

A copy of object suitable for doSim with an extended dataset attached as an attribute named newData.

Examples

```r
fm <- lmer(y ~ x + (1|g), data=simdata)
nrow(example)
fmx1 <- extend(fm, along="x", n=20)
nrow(getData(fmx1))
fmx2 <- extend(fm, along="x", values=c(1,2,4,8,16))
nrow(getData(fmx2))
```
**getData**

Get an object’s data.

---

**Description**

Get the data associated with a model object.

**Usage**

```r
getData(object)
```

```r
getData(object) <- value
```

**Arguments**

- `object`: a fitted model object (e.g., an object of class `merMod` or `lm`).
- `value`: a new `data.frame` to replace the old one. The new data will be stored in the `newData` attribute.

**Details**

Looks for data in the following order:

1. The object’s `newData` attribute, if it has been set by `simr`.
2. The `data` argument of `getCall(object)`, in the environment of `formula(object)`,

**Value**

A `data.frame` with the required data.

**Examples**

```r
lm1 <- lmer(y ~ x + (1|g), data=simdata)
X <- getData(lm1)
```
lastResult

Recover an unsaved simulation

Description
Simulations can take a non-trivial time to run. If the user forgets to assign the result to a variable
this method can recover it.

Usage
lastResult()

See Also
.Last.value

Examples
fm1 <- lmer(y ~ x + (1|g), data=simdata)
powersim(fm1, nsim=10)
ps1 <- lastResult()

makeGlmer

Create an artificial mixed model object

Description
Make a merMod object with the specified structure and parameters.

Usage
makeGlmer(formula, family, fixef, VarCorr, data)
makelmer(formula, fixef, VarCorr, sigma, data)

Arguments
formula a formula describing the model (see glmer).
family type of response variable (see family).
fixef vector of fixed effects
VarCorr variance and covariances for random effects. If there are multiple random ef-
fects, supply their parameters as a list.
data data.frame of explanatory variables.
sigma residual standard deviation.
modify

Modifying model parameters.

Description

These functions can be used to change the size of a model’s fixed effects, its random effect variance/covariance matrices, or its residual variance. This gives you more control over simulations from the model.

Usage

fixef(object) <- value
coef(object) <- value
VarCorr(object) <- value
sigma(object) <- value
scale(object) <- value

Arguments

object a fitted model object.
value new parameter values.

Details

New values for VarCorr are interpreted as variances and covariances, not standard deviations and correlations. New values for sigma and scale are interpreted on the standard deviation scale. This means that both VarCorr(object) <- VarCorr(object) and sigma(object) <- sigma(object) leave object unchanged, as you would expect.
sigma <- will only change the residual standard deviation, whereas scale <- will affect both sigma and VarCorr.

These functions can be used to change the value of individual parameters, such as a single fixed effect coefficient, using standard R subsetting commands.

See Also

getData if you want to modify the model’s data.

Examples

fm <- lmer(y ~ x + (1|g), data=simdata)
fixef(fm)
fixef(fm)["x"] <- -0.1
fixef(fm)
powerCurve

Estimate power at a range of sample sizes.

Description

This function runs powerSim over a range of sample sizes.

Usage

```
powerCurve(fit, test = fixed(getDefaultXname(fit)), sim = fit,
    along = getDefaultXname(fit), within, breaks, seed, fitopts = list(),
    testopts = list(), simopts = list(), ...)
```

Arguments

- **fit**: a fitted model object (see doFit).
- **test**: specify the test to perform. By default, the first fixed effect in fit will be tested. (see: tests).
- **sim**: an object to simulate from. By default this is the same as fit (see doSim).
- **along**: the name of an explanatory variable. This variable will have its number of levels varied.
- **within**: names of grouping variables, separated by "+" or ",". Each combination of groups will be extended to n rows.
- **breaks**: number of levels of the variable specified by along at each point on the power curve.
- **seed**: specify a random number generator seed, for reproducible results.
- **fitopts**: extra arguments for doFit.
- **testopts**: extra arguments for doTest.
- **simopts**: extra arguments for doSim.
- **...**: any additional arguments are passed on to simrOptions. Common options include:
  - **nsim**: the number of simulations to run (default is 1000).
  - **alpha**: the significance level for the statistical test (default is 0.05).
  - **progress**: use progress bars during calculations (default is TRUE).

See Also

- `print.powerCurve`
- `summary.powerCurve`
- `confint.powerCurve`
### powerSim

Estimate power by simulation.

**Description**

Perform a power analysis for a mixed model.

**Usage**

```r
powerSim(fit, test = fixed(getDefaultXname(fit)), sim = fit, fit0pts = list(), test0pts = list(), sim0pts = list(), seed, ...)```

**Arguments**

- `fit`: a fitted model object (see `doFit`).
- `test`: specify the test to perform. By default, the first fixed effect in `fit` will be tested. (see: `tests`).
- `sim`: an object to simulate from. By default this is the same as `fit` (see `doSim`).
- `fit0pts`: extra arguments for `doFit`.
- `test0pts`: extra arguments for `doTest`.
- `sim0pts`: extra arguments for `doSim`.
- `seed`: specify a random number generator seed, for reproducible results.
- `...`: any additional arguments are passed on to `simrOptions`. Common options include:
  - `nsim`: the number of simulations to run (default is 1000).
  - `alpha`: the significance level for the statistical test (default is 0.05).
  - `progress`: use progress bars during calculations (default is TRUE).

**See Also**

`print.powerSim`, `summary.powerSim`, `confint.powerSim`

**Examples**

```r
fm1 <- lmer(y ~ x + (1|g), data=simdata)
powerSim(fm1, nsim=10)
```
print.powerSim

Report simulation results

Description

Describe and extract power simulation results

Usage

```r
## S3 method for class 'powerSim'
print(x, alpha = x$alpha, level = 0.95, ...)

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

## S3 method for class 'powerSim'
summary(object, alpha = object$alpha, level = 0.95,
       method = getSimrOption("binom"), ...)

## S3 method for class 'powerCurve'
summary(object, alpha = object$alpha, level = 0.95,
       method = getSimrOption("binom"), ...)

## S3 method for class 'powerSim'
confint(object, parm, level = 0.95,
       method = getSimrOption("binom"), alpha = object$alpha, ...)

## S3 method for class 'powerCurve'
confint(object, parm, level = 0.95,
       method = getSimrOption("binom"), ...)
```

Arguments

- `x` a `powerSim` or `powerCurve` object
- `alpha` the significance level for the statistical test (default is that used in the call to `powersim`).
- `level` confidence level for power estimate
- `...` additional arguments to pass to `binom::binom.confint()`
- `object` a `powerSim` or `powerCurve` object
- `method` method to use for computing binomial confidence intervals (see `binom::binom.confint()`)
- `parm` currently ignored, included for S3 compatibility with `stats::confint`
**simdata**  
*Example dataset.*

**Description**  
A simple artificial data set used in the tutorial. There are two response variables, a Poisson count \( z \) and a Gaussian response \( y \). There is a continuous predictor \( x \) with ten values \( \{1, 2, \ldots, 10\} \) and a categorical predictor \( g \) with three levels \( \{a, b, c\} \).

**simrOptions**  
*Options Settings for simr*

**Description**  
Control the default behaviour of simr analyses.

**Usage**

`simrOptions(...)`

`getSimrOption(opt)`

**Arguments**

- `...`: a list of names to get options, or a named list of new values to set options.
- `opt`: option name (character string).

**Value**

- `getSimrOption` returns the current value for the option \( x \).
- `simrOptions` returns
  1. a named list of all options, if no arguments are given.
  2. a named list of specified options, if a list of option names is given.
  3. (invisibly) a named list of changed options with their previous values, if options are set.

**See Also**

`binom::binom.confint, powersim, powerCurve`
**Options in simr**

Options that can be set with this method (and their default values).

- `nsim` default number of simulations (1000).
- `alpha` default confidence level (0.05).
- `progress` use progress bars during calculations (TRUE).
- `binom` method for calculating confidence intervals ("exact").
- `pbnsim` number of simulations for parametric bootstrap tests using pbkrtest (100).
- `pcmin` minimum number of levels for the smallest point on a `powerCurve` (3).
- `pcmax` maximum number of points on the default `powerCurve` (10).
- `observedPowerWarning` warn if an unmodified fitted model is used (TRUE).
- `carTestType` type of test, i.e. type of sum of squares, for tests performed with `car::anova` ("II").
- `lmerTestDf` approximation to use for denominator degrees of freedom for tests performed with `lmerTest` ("Satterthwaite"). Note that setting this option to "lme4" will reduce the `lmerTest` model to an `lme4` model and break functionality based on `lmerTest`.
- `lmerTestType` type of test, i.e. type of sum of squares, for F-tests performed with `lmerTest::anova.lmerModLmerTest` (2). Note that unlike the tests performed with `car::anova`, the test type must be given as a number and not a character.

**Examples**

```r
getSimrOption("nsim")
oldopts <- simrOptions(nsim=5)
getSimrOption("nsim")
simrOptions(oldopts)
getSimrOption("nsim")
```

---

**Description**

Specify a statistical test to apply

**Usage**

- `fixed(xname, method = c("z", "t", "f", "chisq", "anova", "lr", "sa", "kr", "pb"))`
- `compare(model, method = c("lr", "pb"))`
- `fcompare(model, method = c("lr", "kr", "pb"))`
rcompare(model, method = c("lr", "pb"))
random()

Arguments

- xname: an explanatory variable to test (character).
- method: the type of test to apply (see Details).
- model: a null model for comparison (formula).

Details

- fixed: Test a single fixed effect, specified by xname.
- compare: Compare the current model to a smaller one specified by the formula model.
- fcompare, rcompare: Similar to compare, but only the fixed/random part of the formula needs to be supplied.
- random: Test the significance of a single random effect.

Value

A function which takes a fitted model as an argument and returns a single p-value.

Methods

The method argument can be used to specify one of the following tests. Note that "z" is an asymptotic approximation for models not fitted with glmer and "kr" will only work with models fitted with lmer.

- z: Z-test for models fitted with glmer (or glm), using the p-value from summary. For models fitted with lmer, this test can be used to treat the t-values from summary as z-values, which is equivalent to assuming infinite degrees of freedom. This asymptotic approximation seems to perform well for even medium-sized data sets, as the denominator degrees of freedom are already quite large (cf. Baayen et al. 2008) even if calculating their exact value is analytically unsolved and computationally difficult (e.g. with Satterthwaite or Kenward-Roger approximations). Setting alpha=0.045 is roughly equal to the t=2 threshold suggested by Baayen et al. (2008) and helps compensate for the slightly anti-conservative approximation.
- t: T-test for models fitted with lm. Also available for mixed models when lmerTest is installed, using the p-value calculated using the Satterthwaite approximation for the denominator degrees of freedom by default. This can be changed by setting lmerTestDdf, see simrOptions.
- lr: Likelihood ratio test, using anova.
- f: Wald F-test, using car::Anova. Useful for examining categorical terms. For models fitted with lmer, this should yield equivalent results to method='kr'. Uses Type-II tests by default, this can be changed by setting carTestType, see simrOptions.
- chisq: Wald Chi-Square test, using car::Anova. Please note that while this is much faster than the F-test computed with Kenward-Roger, it is also known to be anti-conservative, especially for small samples. Uses Type-II tests by default, this can be changed by setting carTestType, see simrOptions.
anova: ANOVA-style F-test, using `anova` and `lmerTest::anova.lmerModLmerTest`. For `lm`, this yields a Type-I (sequential) test (see `anova`); to use other test types, use the F-tests provided by `car::Anova()` (see above). For `lmer`, this generates Type-II tests with Satterthwaite denominator degrees of freedom by default, this can be changed by setting `lmerTestDdf` and `lmerTestType`, see `simrOptions`.

kr: Kenward-Roger test, using `KRmodcomp`. This only applies to models fitted with `lmer`, and compares models with different fixed effect specifications but equivalent random effects.

pb: Parametric bootstrap test, using `PBmodcomp`. This test will be very accurate, but is also very computationally expensive.

Tests using `random` for a single random effect call `exactRLRT`.

References


Examples

```r
lm1 <- lmer(y ~ x + (x|g), data=simdata)  # mixed-effects model
lm0 <- lmer(y ~ x + (1|g), data=simdata)  # constant effect
anova(lm1, lm0)
compare(~ x + (1|g))(lm1)
rcompare(~ (1|g))(lm1)
## Not run: powerSim(fm1, compare(~ x + (1|g)))
```
Index

.Last.value, 6
anova, 13, 14
binom::binom.confint, 11
binom::binom.confint(), 10
car::Anova, 12, 13
coeff<- (modify), 7
compare (tests), 12
confint.powerCurve, 8
confint.powerCurve (print.powerSim), 10
confint.powerSim, 9
confint.powerSim (print.powerSim), 10
doFit, 2, 3, 8, 9
doSim, 3, 4, 8, 9
doTest, 3, 8, 9
effectRLRT, 14
extend, 4
family, 6
fcompare (tests), 12
fixed (tests), 12
fixef<- (modify), 7
data, 5, 7
data<- (getData), 5
getSimrOption (simrOptions), 11
glm, 13
glmer, 6, 13
PBmodcomp, 14
powerCurve, 2, 8, 10–12
powerSim, 8, 9, 10, 11
print.powerCurve, 8
print.powerCurve (print.powerSim), 10
print.powerSim, 9, 10
random (tests), 12
rcompare (tests), 12
scale<- (modify), 7
sigma<- (modify), 7
simdata, 11
simr-package, 2
simrOptions, 8, 9, 11, 13, 14
stats::confint, 10
summary, 13
summary.powerCurve, 8
summary.powerCurve (print.powerSim), 10
summary.powerSim, 9
summary.powerSim (print.powerSim), 10
tests, 3, 8, 9, 12
VarCorr<- (modify), 7

makeLmer (makeGlmer), 6
merMod, 6
modify, 7

15