Package 'Spower'

September 5, 2025

Title Power Analyses using Monte Carlo Simulations

Version 0.4.0

Description Provides a general purpose simulation-based power analysis API for routine and customized simulation experimental designs. The package focuses exclusively on Monte Carlo simulation experiment variants of (expected) prospective power analyses, criterion analyses, compromise analyses, sensitivity analyses, and a priori/post-hoc analyses. The default simulation experiment functions defined within the package provide stochastic variants of the power analysis subroutines in G*Power 3.1 (Faul, Erdfelder, Buchner, and Lang, 2009) <doi:10.3758/brm.41.4.1149>, along with various other parametric and non-parametric power analysis applications (e.g., mediation analyses) and support for Bayesian power analysis by way of Bayes factors or posterior probability evaluations. Additional functions for building empirical power curves, reanalyzing simulation information, and for increasing the precision of the resulting power estimates are also included, each of which utilize similar API structures.

Depends SimDesign (>= 2.20.0), R (>= 4.1.0), stats

Imports cocor, car, polycor, parallelly, methods, ggplot2, plotly, lavaan, EnvStats

Suggests knitr, rmarkdown, bookdown, VGAM, copula, pwr

VignetteBuilder knitr

License GPL (>= 3)

Encoding UTF-8

Repository CRAN

Maintainer Phil Chalmers <rphilip.chalmers@gmail.com>

URL https://github.com/philchalmers/Spower

BugReports https://github.com/philchalmers/Spower/issues?state=open

RoxygenNote 7.3.2

NeedsCompilation no

Author Phil Chalmers [aut, cre] (ORCID:
<https://orcid.org/0000-0001-5332-2810>)

Date/Publication 2025-09-05 20:00:02 UTC

Contents

`getLastSpower` *Get previously evaluated Spower execution*

Description

If the result of `Spower` or `SpowerBatch` was not stored into an object this function will retrieve the last evaluation.

Usage

```
getLastSpower()
```

Value

the last object returned from `Spower` or `SpowerBatch`

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

`is.CI_within` *Evaluate whether a confidence interval is within a tolerable interval*

Description

Return `TRUE` if an estimated confidence interval falls within a tolerable `interval` range. Typically used for equivalence, superiority, or non-inferiority testing.

Usage

```
is.CI_within(CI, interval)
```

Arguments

`CI`	estimated confidence interval (length 2)
`interval`	tolerable interval range (length 2)

Value

logical

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

`is.outside_CI`, `Spower`

Examples

```
CI <- c(.2, .4)
LU <- c(.1, .3)
is.CI_within(CI, LU)        # not within tolerable interval
is.CI_within(CI, c(0, .5))  # is within wider interval

# complement indicates if CI is outside interval
!is.CI_within(CI, LU)

#####
# for superiority test
is.CI_within(CI, c(.1, Inf))  # CI is within tolerable interval

# for inferiority test
is.CI_within(CI, c(-Inf, .3))  # CI is not within tolerable interval
```

`is.outside_CI` *Evaluate whether parameter is outside a given confidence interval*

Description

Returns `TRUE` if parameter reflecting a null hypothesis falls outside a given confidence interval. This is an alternative approach to writing an experiment that returns a p-value.

Usage

```
is.outside_CI(P0, CI)
```

Arguments

`P0`	parameter to evaluate
`CI`	confidence interval

Value

logical

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

`is.CI_within`, `Spower`

Examples

```
p0 <- .3
CI <- c(.2, .4)
is.outside_CI(p0, CI)

# complement indicates if p0 is within CI
!is.outside_CI(p0, CI)
```

p_2r *p-value from comparing two or more correlations simulation*

Description

Function utilizes cocor to perform correlation comparison for independent, overlapping, and non-overlapping designs.

Usage

```
p_2r(
  n,
  r.ab1,
  r.ab2,
  r.ac1,
  r.ac2,
  r.bc1,
  r.bc2,
  r.ad1,
  r.ad2,
  r.bd1,
  r.bd2,
  r.cd1,
  r.cd2,
  n2_n1 = 1,
  two.tailed = TRUE,
  type = c("independent", "overlap", "nonoverlap"),
  test = "fisher1925",
  gen_fun = gen_2r,
  ...
)

gen_2r(n, R, ...)
```

Arguments

n	sample size
r.ab1	correlation between variable A and B in sample 1
r.ab2	correlation between variable A and B in sample 2
r.ac1	same pattern as r.ab1
r.ac2	same pattern as r.ab2
r.bc1	...
r.bc2	...
r.ad1	...
r.ad2	...

r.bd1	...
r.bd2	...
r.cd1	...
r.cd2	...
n2_n1	sample size ratio
two.tailed	logical; use two-tailed test?
type	type of correlation design
test	hypothesis method to use. Defaults to 'fisher1925'
gen_fun	function used to generate the required discrete data. Object returned must be a matrix with n rows. Default uses gen_2r. User defined version of this function must include the argument ...
...	additional arguments to be passed to gen_fun. Not used unless a customized gen_fun is defined
R	a correlation matrix constructed from the inputs to p_2r

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

Examples

```
# independent (same x-y pairing across groups)
p_2r(100, r.ab1=.5, r.ab2=.6)



   # estimate empirical power
   p_2r(n=100, r.ab1=.5, r.ab2=.6) |> Spower()

   # estimate n required to reach 80% power
   p_2r(n=NA, r.ab1=.5, r.ab2=.6) |>
        Spower(power=.80, interval=c(100, 5000))



# overlap (same y, different xs)
p_2r(100, r.ab1=.5, r.ab2=.7,
          r.ac1=.3, r.ac2=.3,
          r.bc1=.2, r.bc2=.2, type = 'overlap')

# nonoverlap (different ys, different xs)
p_2r(100, r.ab1=.5, r.ab2=.6,
          r.ac1=.3, r.ac2=.3,
          r.bc1=.2, r.bc2=.2,
```

```
            r.ad1=.2, r.ad2=.2,
            r.bd1=.4, r.bd2=.4,
            r.cd1=.2, r.cd2=.2,
            type = 'nonoverlap')
```

`p_anova.test` *p-value from one-way ANOVA simulation*

Description

Generates continuous multi-sample data to be analyzed by a one-way ANOVA, and return a p-value. Uses the function `oneway.test` to perform the analyses. The data and associated test assume that the conditional observations are normally distributed and have have equal variance by default, however these may be modified.

Usage

```
p_anova.test(
  n,
  k,
  f,
  n.ratios = rep(1, k),
  two.tailed = TRUE,
  var.equal = TRUE,
  means = NULL,
  sds = NULL,
  gen_fun = gen_anova.test,
  ...
)

gen_anova.test(n, k, f, n.ratios = rep(1, k), means = NULL, sds = NULL, ...)
```

Arguments

`n`	sample size per group
`k`	number of groups
`f`	Cohen's f effect size
`n.ratios`	allocation ratios reflecting the sample size ratios. Default of 1 sets the groups to be the same size (n * n.ratio)
`two.tailed`	logical; should a two-tailed or one-tailed test be used?
`var.equal`	logical; use the pooled SE estimate instead of the Welch correction for unequal variances?
`means`	(optional) vector of means. When specified the input `f` is ignored

sds	(optional) vector of SDs. When specified the input f is ignored
gen_fun	function used to generate the required data. Object returned must be a matrix with k rows and k columns of numeric data. Default uses gen_anova.test. User defined version of this function must include the argument ...
...	additional arguments to be passed to gen_fun. Not used unless a customized gen_fun is defined

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

gen_anova.test

Examples

```
# n=50 in 3 groups, "medium" effect size
p_anova.test(50, k=3, f=.25)

# explicit means/sds
p_anova.test(50, 3, means=c(0,0,1), sds=c(1,2,1))


  # compare simulated results to pwr package
  pwr::pwr.anova.test(f=0.28, k=4, n=20)
  p_anova.test(n=20, k=4, f=.28) |> Spower()
```

---

p_chisq.test *p-value from chi-squared test simulation*

Description

Generates multinomial data suitable for analysis with chisq.test.

Usage

```
p_chisq.test(
  n,
  w,
  df,
  correct = TRUE,
```

```
  P0 = NULL,
  P = NULL,
  gen_fun = gen_chisq.test,
  ...
)

gen_chisq.test(n, P, ...)
```

Arguments

n	sample size per group
w	Cohen's w effect size
df	degrees of freedom
correct	logical; apply continuity correction?
P0	specific null pattern, specified as a numeric vector or matrix
P	specific power configuration, specified as a numeric vector or matrix
gen_fun	function used to generate the required discrete data. Object returned must be a matrix with k rows and k columns of counts. Default uses gen_chisq.test. User defined version of this function must include the argument ...
...	additional arguments to be passed to gen_fun. Not used unless a customized gen_fun is defined

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

gen_chisq.test

Examples

```
# effect size w + df
p_chisq.test(100, w=.2, df=3)

# vector of explicit probabilities (goodness of fit test)
p_chisq.test(100, P0 = c(.25, .25, .25, .25),
                  P = c(.6, .2, .1, .1))

# matrix of explicit probabilities (two-dimensional test of independence)
p_chisq.test(100, P0 = matrix(c(.25, .25, .25, .25), 2, 2),
                  P = matrix(c(.6, .2, .1, .1),2,2))
```

```
# compare simulated results to pwr package

P0 <- c(1/3, 1/3, 1/3)
P <- c(.5, .25, .25)
w <- pwr::ES.w1(P0, P)
df <- 3-1
pwr::pwr.chisq.test(w=w, df=df, N=100, sig.level=0.05)

# slightly less power when evaluated empirically
p_chisq.test(n=100, w=w, df=df) |> Spower(replications=100000)
p_chisq.test(n=100, P0=P0, P=P) |> Spower(replications=100000)

# slightly differ (latter more conservative due to finite sampling behaviour)
pwr::pwr.chisq.test(w=w, df=df, power=.8, sig.level=0.05)
p_chisq.test(n=NA, w=w, df=df) |>
       Spower(power=.80, interval=c(50, 200))
p_chisq.test(n=NA, w=w, df=df, correct=FALSE) |>
       Spower(power=.80, interval=c(50, 200))

# Spower slightly more conservative even with larger N
pwr::pwr.chisq.test(w=.1, df=df, power=.95, sig.level=0.05)
p_chisq.test(n=NA, w=.1, df=df) |>
       Spower(power=.95, interval=c(1000, 2000))
p_chisq.test(n=NA, w=.1, df=df, correct=FALSE) |>
       Spower(power=.95, interval=c(1000, 2000))
```

`p_glm` *p-value from (generalized) linear regression model simulations with fixed predictors*

Description

p-values associated with (generalized) linear regression model. Requires a pre-specified design matrix (`X`).

Usage

```
p_glm(
  formula,
  X,
  betas,
  test,
  sigma = NULL,
  family = gaussian(),
  gen_fun = gen_glm,
  ...
```

```
)

gen_glm(formula, X, betas, sigma = NULL, family = gaussian(), ...)
```

Arguments

formula	formula passed to either lm or glm
X	a data.frame containing the covariates
betas	vector of slope coefficients that match the model.matrix version of X
test	character vector specifying the test to pass to lht. Can also be a list of character vectors to evaluate multiple tests
sigma	residual standard deviation for linear model. Only used when family = 'gaussian'
family	family of distributions to use (see family)
gen_fun	function used to generate the required discrete data. Object returned must be a data.frame. Default uses gen_glm. User defined version of this function must include the argument ...
...	additional arguments to be passed to gen_fun. Not used unless a customized gen_fun is defined

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

p_lm.R2

Examples

```
X <- data.frame(G = factor(rep(c('control', 'treatment'), each=50)),
                C = sample(50:100, 100, replace=TRUE))
head(X)

# ANCOVA setup
p_glm(y ~ G + C, test="Gtreatment = 0",
  X=X, betas=c(10, .3, 1), sigma=1)

# ANCOVA setup with logistic regression
p_glm(y ~ G + C, test="Gtreatment = 0",
  X=X, betas=c(-2, .5, .01), family=binomial())

# ANCOVA setup with poisson regression
p_glm(y ~ G + C, test="Gtreatment = 0",
  X=X, betas=c(-2, .5, .01), family=poisson())
```

```
# test whether two slopes differ given different samples.
#   To do this setup data as an MLR where a binary variable S
#   is used to reflect the second sample, and the interaction
#   effect evaluates the magnitude of the slope difference
gen_twogroup <- function(n, dbeta, sdx1, sdx2, sigma, n2_n1 = 1, ...){
  X1 <- rnorm(n, sd=sdx1)
  X2 <- rnorm(n*n2_n1, sd=sdx2)
  X <- c(X1, X2)
  N <- length(X)
  S <- c(rep(0, n), rep(1, N-n))
  y <- dbeta * X*S + rnorm(N, sd=sigma)
  dat <- data.frame(y, X, S)
  dat
}

# prospective power using test that interaction effect is equal to 0
p_glm(formula=y~X*S, test="X:S = 0",
  n=100, sdx1=1, sdx2=2, dbeta=0.2,
  sigma=0.5, gen_fun=gen_twogroup) |> Spower(replications=1000)
```

`p_kruskal.test` *p-value from Kruskal-Wallis Rank Sum Test simulation*

Description

Simulates data given two or more parent distributions and returns a p-value using `kruskal.test`. Default generates data from Gaussian distributions, however this can be modified.

Usage

```
p_kruskal.test(
  n,
  k,
  means,
  n.ratios = rep(1, k),
  gen_fun = gen_kruskal.test,
  ...
)

gen_kruskal.test(n, k, n.ratios, means, ...)
```

Arguments

n	sample size per group
k	number of groups
means	vector of means to control location parameters
n.ratios	allocation ratios reflecting the sample size ratios. Default of 1 sets the groups to be the same size (n * n.ratio)
gen_fun	function used to generate the required data. Object returned must be a `list` of length k, where each element contains the sample data in each group. Default uses `gen_kruskal.test`. User defined version of this function must include the argument `...`
...	additional arguments to pass to gen_fun

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

Examples

```
# three group test where data generate from Gaussian distributions
p_kruskal.test(n=30, k=3, means=c(0, .5, .6))

# generate data from chi-squared distributions with different variances
gen_chisq <- function(n, k, n.ratios, means, dfs, ...){
  dat <- vector('list', k)
  ns <- n * n.ratios
  for(g in 1:k)
  dat[[g]] <- rchisq(ns[g], df=dfs[g]) - dfs[g] + means[g]
  dat
}

p_kruskal.test(n=30, k=3, means=c(0, 1, 2),
   gen_fun=gen_chisq, dfs=c(10, 15, 20))


  # empirical power estimate
  p_kruskal.test(n=30, k=3, means=c(0, .5, .6)) |> Spower()
  p_kruskal.test(n=30, k=3, means=c(0, 1, 2), gen_fun=gen_chisq,
         dfs = c(10, 15, 20)) |> Spower()
```

p_ks.test *p-value from Kolmogorov-Smirnov one- or two-sample simulation*

Description

Generates one or two sets of continuous data group-level data and returns a p-value under the null that the groups were drawn from the same distribution (two sample) or from a theoretically known distribution (one sample).

Usage

```
p_ks.test(n, p1, p2, n2_n1 = 1, two.tailed = TRUE, parent = NULL, ...)
```

Arguments

n	sample size per group, assumed equal across groups
p1	a function indicating how the data were generated for group 1
p2	(optional) a function indicating how the data were generated for group 2. If omitted a one-sample test will be evaluated provided that parent is also specified
n2_n1	sample size ratio. Default uses equal sample sizes
two.tailed	logical; should a two-tailed or one-tailed test be used?
parent	the cumulative distribution function to use (e.g., pnorm). Specifying this input will construct a one-sample test setup
...	additional arguments to be passed to the parent distribution function from ks.test, as well as any other relevant parameter to ks.test (e.g., exact = TRUE)

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

gen_t.test

Examples

```
# two-sample test from two Gaussian distributions with different locations
p1 <- function(n) rnorm(n)
p2 <- function(n) rnorm(n, mean=-.5)
p_ks.test(n=100, p1, p2)

# one-sample data from chi-squared distribution tested
#   against a standard normal distribution
pc <- function(n, df=15) (rchisq(n, df=df) - df) / sqrt(2*df)
p_ks.test(n=100, p1=pc, parent=pnorm, mean=0, sd=1)


  # empirical power estimates
  p_ks.test(n=100, p1, p2) |> Spower()
  p_ks.test(n=100, p1=pc, parent=pnorm, mean=0, sd=1) |> Spower()
```

---

p_lm.R2 *p-value from global linear regression model simulation*

Description

p-values associated with linear regression model using fixed/random independent variables. Focus is on the omnibus behavior of the R^2 statistic.

Usage

```
p_lm.R2(n, R2, k, R2_0 = 0, k.R2_0 = 0, R2.resid = 1 - R2, fixed = TRUE, ...)
```

Arguments

n	sample size
R2	R-squared effect size
k	number of IVs
R2_0	null hypothesis for R-squared
k.R2_0	number of IVs associated with the null hypothesis model
R2.resid	residual R-squared value, typically used when comparing nested models when fit sequentially (e.g., comparing model A vs B when model involves the structure A -> B -> C)
fixed	logical; if FALSE then the data are random generated according to a joint multivariate normal distribution
...	additional arguments to be passed to gen_fun. Not used unless a customized gen_fun is defined

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

p_glm

Examples

```
# 5 fixed IVs, R^2 = .1, sample size of 95
p_lm.R2(n=95, R2=.1, k=5)

# random model
p_lm.R2(n=95, R2=.1, k=5, fixed=FALSE)
```

---

p_mauchly.test *p-value from Mauchly's Test of Sphericity simulation*

Description

Perform simulation experiment for Mauchly's Test of Sphericity using the function `mauchlys.test`, returning a p-value. Assumes the data are from a multivariate normal distribution, however this can be modified.

Usage

```
p_mauchly.test(n, sigma, gen_fun = gen_mauchly.test, ...)

gen_mauchly.test(n, sigma, ...)

mauchlys.test(X)
```

Arguments

n	sample size
sigma	symmetric covariance/correlation matrix passed to gen_fun
gen_fun	function used to generate the required data. Object returned must be a matrix with K columns and n rows. Default uses gen_mauchly.test to generate multivariate normal samples. User defined version of this function must include the argument ...
...	additional arguments to be passed to gen_fun. Not used unless a customized gen_fun is defined
X	a matrix with k columns and n rows

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

Examples

```
sigma <- diag(c(1,2,1))
sigma

p_mauchly.test(100, sigma=sigma)

# Null is true
sigma.H0 <- diag(3)
p_mauchly.test(100, sigma=sigma.H0)



    # empirical power estimate
    p_mauchly.test(100, sigma=sigma) |> Spower()

    # empirical Type I error estimate
    p_mauchly.test(100, sigma=sigma.H0) |> Spower()
```

---

p_mcnemar.test *p-value from McNemar test simulation*

Description

Generates two-dimensional sample data for McNemar test and return a p-value. Uses `mcnemar.test`.

Usage

```
p_mcnemar.test(
  n,
  prop,
  two.tailed = TRUE,
  correct = TRUE,
  gen_fun = gen_mcnemar.test,
  ...
)

gen_mcnemar.test(n, prop, ...)
```

Arguments

n	total sample size
prop	two-dimensional matrix of proportions/probabilities
two.tailed	logical; should a two-tailed or one-tailed test be used?
correct	logical; use continuity correction? Only applicable for 2x2 tables
gen_fun	function used to generate the required discrete data. Object returned must be a matrix with k rows and k columns of counts. Default uses gen_mcnemar.test. User defined version of this function must include the argument ...
...	additional arguments to be passed to gen_fun. Not used unless a customized gen_fun is defined

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

gen_mcnemar.test

Examples

```
# from ?mcnemar.test
Performance <- matrix(c(794, 86, 150, 570),
   nrow = 2,
   dimnames = list("1st Survey" = c("Approve", "Disapprove"),
               "2nd Survey" = c("Approve", "Disapprove")))
(prop <- prop.table(Performance))

# one sample + test and resulting p-value
p_mcnemar.test(n=sum(Performance), prop=prop)



# post-hoc power (not recommended)
Spower(p_mcnemar.test(n=sum(Performance), prop=prop))
```

p_mediation *p-value from three-variable mediation analysis simulation*

Description

Simple 3-variable mediation analysis simulation to test the hypothesis that X -> Y is mediated by the relationship X -> M -> Y. Currently, M and Y are assumed to be continuous variables with Gaussian errors, while X may be continuous or dichotomous.

Usage

```
p_mediation(
  n,
  a,
  b,
  cprime,
  dichotomous.X = FALSE,
  two.tailed = TRUE,
  method = "wald",
  sd.X = 1,
  sd.Y = 1,
  sd.M = 1,
  gen_fun = gen_mediation,
  ...
)

gen_mediation(
  n,
  a,
  b,
  cprime,
  dichotomous.X = FALSE,
  sd.X = 1,
  sd.Y = 1,
  sd.M = 1,
  ...
)
```

Arguments

n	total sample size unless dichotomous.X = TRUE, in which the value represents the size per group
a	regression coefficient for the path X -> M
b	regression coefficient for the path M -> Y
cprime	partial regression coefficient for the path X -> Y

dichotomous.X	logical; should the X variable be generated as though it were dichotomous? If TRUE then n represents the sample size per group
two.tailed	logical; should a two-tailed or one-tailed test be used?
method	type of inferential method to use. Default uses the Wald (a.k.a., Sobel) test
sd.X	standard deviation for X
sd.Y	standard deviation for Y
sd.M	standard deviation for M
gen_fun	function used to generate the required two-sample data. Object returned must be a data.frame with the columns "DV" and "group". Default uses gen_mediation to generate conditionally Gaussian distributed samples. User defined version of this function must include the argument ...
...	additional arguments to be passed to gen_fun. Not used unless a customized gen_fun is defined

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

gen_mediation

Examples

```
# joint test H0: a*b = 0
p_mediation(50, a=sqrt(.35), b=sqrt(.35), cprime=.39)
p_mediation(50, a=sqrt(.35), b=sqrt(.35), cprime=.39, dichotomous.X=TRUE)



  # power to detect mediation
  p_mediation(n=50, a=sqrt(.35), b=sqrt(.35), cprime=.39) |>
    Spower(parallel=TRUE, replications=1000)

  # sample size estimate for .95 power
  p_mediation(n=NA, a=sqrt(.35), b=sqrt(.35), cprime=.39) |>
    Spower(power=.95, interval=c(50, 200), parallel=TRUE)
```

p_prop.test *p-value from proportion test simulation*

Description

Generates single and multi-sample data for proportion tests and return a p-value. Uses `binom.test` for one-sample applications and `prop.test` otherwise.

Usage

```
p_prop.test(
  n,
  h,
  prop = NULL,
  pi = 0.5,
  n.ratios = rep(1, length(prop)),
  two.tailed = TRUE,
  correct = TRUE,
  exact = FALSE,
  gen_fun = gen_prop.test,
  ...
)

gen_prop.test(
  n,
  h,
  prop = NULL,
  pi = 0.5,
  n.ratios = rep(1, length(prop)),
  ...
)
```

Arguments

n	sample size per group
h	Cohen's h effect size; only supported for one-sample analysis. Note that it's important to specify the null value `pi` when supplying this effect size as the power changes depending on these specific values (see example below).
prop	sample probability/proportions of success. If a vector with two-values or more elements are supplied then a multi-samples test will be used. Matrices are also supported
pi	probability of success to test against (default is .5). Ignored for two-sample tests
n.ratios	allocation ratios reflecting the sample size ratios. Default of 1 sets the groups to be the same size (n * n.ratio)

two.tailed	logical; should a two-tailed or one-tailed test be used?
correct	logical; use Yates' continuity correction?
exact	logical; use fisher's exact test via fisher.test? Use of this flag requires that prop was specified as a matrix
gen_fun	function used to generate the required discrete data. Object returned must be a matrix with two rows and 1 or more columns. Default uses gen_prop.test. User defined version of this function must include the argument ...
...	additional arguments to be passed to gen_fun. Not used unless a customized gen_fun is defined

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

gen_prop.test

Examples

```
# one sample, 50 observations, tested against pi = .5 by default
p_prop.test(50, prop=.65)

# specified using h and pi
h <- pwr::ES.h(.65, .4)
p_prop.test(50, h=h, pi=.4)
p_prop.test(50, h=-h, pi=.65)

# two-sample test
p_prop.test(50, prop=c(.5, .65))

# two-sample test, unequal ns
p_prop.test(50, prop=c(.5, .65), n.ratios = c(1,2))

# three-sample test, group2 twice as large as others
p_prop.test(50, prop=c(.5, .65, .7), n.ratios=c(1,2,1))

# Fisher exact test
p_prop.test(50, prop=matrix(c(.5, .65, .7, .5), 2, 2))


    # compare simulated results to pwr package

    # one-sample tests
    (h <- pwr::ES.h(0.5, 0.4))
    pwr::pwr.p.test(h=h, n=60)
```

```
# uses binom.test (need to specify null location as this matters!)
Spower(p_prop.test(n=60, h=h, pi=.4))
Spower(p_prop.test(n=60, prop=.5, pi=.4))

# compare with switched null
Spower(p_prop.test(n=60, h=h, pi=.5))
Spower(p_prop.test(n=60, prop=.4, pi=.5))

# two-sample test, one-tailed
(h <- pwr::ES.h(0.67, 0.5))
pwr::pwr.2p.test(h=h, n=80, alternative="greater")
p_prop.test(n=80, prop=c(.67, .5), two.tailed=FALSE,
  correct=FALSE) |> Spower()

# same as above, but with continuity correction (default)
p_prop.test(n=80, prop=c(.67, .5), two.tailed=FALSE) |>
  Spower()

# three-sample joint test, equal n's
p_prop.test(n=50, prop=c(.6,.4,.7)) |> Spower()
```

---

p_r *p-value from correlation simulation*

Description

Generates correlated X-Y data and returns a p-value to assess the null of no correlation in the population. The X-Y data are generated assuming a bivariate normal distribution.

Usage

```
p_r(n, r, rho = 0, method = "pearson", two.tailed = TRUE, gen_fun = gen_r, ...)

gen_r(n, r, ...)
```

Arguments

n	sample size
r	correlation
rho	population coefficient to test against. Uses the Fisher's z-transformation approximation when non-zero
method	method to use to compute the correlation (see cor.test). Only used when rho = 0
two.tailed	logical; should a two-tailed or one-tailed test be used?

gen_fun — function used to generate the required dependent bivariate data. Object returned must be a `matrix` with two columns and `n` rows. Default uses `gen_r` to generate conditionally dependent data from a bivariate normal distribution. User defined version of this function must include the argument `...`

... — additional arguments to be passed to `gen_fun`. Not used unless a customized `gen_fun` is defined

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

`gen_r`

Examples

```
# 50 observations, .5 correlation
p_r(50, r=.5)
p_r(50, r=.5, method = 'spearman')

# test against constant other than rho = .6
p_r(50, .5, rho=.60)



    # compare simulated results to pwr package

    pwr::pwr.r.test(r=0.3, n=50)
    p_r(n=50, r=0.3) |> Spower()

    pwr::pwr.r.test(r=0.3, power=0.80)
    p_r(n=NA, r=0.3) |> Spower(power=.80, interval=c(10, 200))

    pwr::pwr.r.test(r=0.1, power=0.80)
    p_r(n=NA, r=0.1) |> Spower(power=.80, interval=c(200, 1000))
```

p_r.cat *p-value from tetrachoric/polychoric or polyserial*

Description

Generates correlated X-Y data and returns a p-value to assess the null of no correlation in the population. The X-Y data are generated assuming a multivariate normal distribution and subsequently discretized for one or both of the variables.

Usage

```
p_r.cat(
  n,
  r,
  tauX,
  rho = 0,
  tauY = NULL,
  ML = TRUE,
  two.tailed = TRUE,
  score = FALSE,
  gen_fun = gen_r,
  ...
)
```

Arguments

n	sample size
r	correlation prior to the discretization (recovered via the polyserial/polychoric estimates)
tauX	intercept parameters used for discretizing the X variable
rho	population coefficient to test against
tauY	intercept parameters used for discretizing the Y variable. If missing a polyserial correlation will be estimated, otherwise a tetrachoric/polychoric correlation will be estimated
ML	logical; use maximum-likelihood estimation?
two.tailed	logical; should a two-tailed or one-tailed test be used?
score	logical; should the SE be based at the null hypothesis (score test) or the ML estimate (Wald test)? The former is the canonical form for a priori power analyses though requires twice as many computations as the Wald test approach
gen_fun	function used to generate the required continuous bivariate data (prior to truncation). Object returned must be a `matrix` with two columns. Default uses `gen_r` to generate conditionally dependent data from a bivariate normal distribution. User defined version of this function must include the argument `...`
...	additional arguments to be passed to `gen_fun`. Not used unless a customized `gen_fun` is defined

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

gen_r

Examples

```
# 100 observations, .5 correlation, tetrachoric estimate
p_r.cat(100, r=.5, tauX=0, tauY=1)

# Wald test
p_r.cat(100, r=.5, tauX=0, tauY=1, score=FALSE)

# polyserial estimate (Y continuous)
p_r.cat(50, r=.5, tauX=0)
```

---

p_scale *p-value from Scale Test simulation*

Description

Simulates data given one or two parent distributions and returns a p-value testing that the scale of the type distributions are the same. Default implementation uses Gaussian distributions, however the distribution function may be modified to reflect other populations of interest. Uses ansari.test or mood.test for the analysis.

Usage

```
p_scale(
  n,
  scale,
  n2_n1 = 1,
  two.tailed = TRUE,
  exact = NULL,
  test = "Ansari",
  parent = function(n, ...) rnorm(n),
  ...
)
```

Arguments

n	sample size per group
scale	the scale to multiply the second group by (1 reflects equal scaling)
n2_n1	sample size ratio
two.tailed	logical; use two-tailed test?
exact	a logical indicating whether an exact p-value should be computed
test	type of method to use. Can be either 'Ansari' or 'Mood'
parent	data generation function (default assumes Gaussian shape). Must be population mean centered
...	additional arguments to pass to simulation functions (if used)

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

Examples

```
# n=30 per group,
#  Distributions Gaussian with sd=1 for first group and sd=2 for second
p_scale(30, scale=2)
p_scale(30, scale=2, test='Mood')

# compare chi-squared distributions
parent <- function(n, df, ...) rchisq(n, df=df) - df
p_scale(30, scale=2, parent=parent, df=3)


  # empirical power of the experiments
  p_scale(30, scale=2) |> Spower()
  p_scale(30, scale=2, test='Mood') |> Spower()

  p_scale(30, scale=2, parent=parent, df=3) |> Spower()
  p_scale(30, scale=2, test='Mood', parent=parent, df=3) |> Spower()
```

p_shapiro.test *p-value from Shapiro-Wilk Normality Test simulation*

Description

Generates univariate distributional data and returns a p-value to assess the null that the population follows a Gaussian distribution shape. Uses shapiro.test.

Usage

```
p_shapiro.test(dist)
```

Arguments

dist	expression used to generate the required sample data

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

Examples

```
# 50 observations drawn from normal distribution (null is true)
p_shapiro.test(rnorm(50))

# 50 observations from slightly skewed chi-squared distribution (power)
p_shapiro.test(rchisq(50, df=100))


    # empirical Type I error rate estimate
    p_shapiro.test(rnorm(50)) |> Spower()

    # power
    p_shapiro.test(rchisq(50, df=100)) |> Spower()
```

p_t.test *p-value from independent/paired samples t-test simulation*

Description

Generates one or two sets of continuous data group-level data according to Cohen's effect size 'd', and returns a p-value. The data and associated t-test assume that the conditional observations are normally distributed and have have equal variance by default, however these may be modified.

Usage

```
p_t.test(
  n,
  d,
  mu = 0,
  r = NULL,
  type = c("two.sample", "one.sample", "paired"),
  n2_n1 = 1,
  two.tailed = TRUE,
  var.equal = TRUE,
  means = NULL,
  sds = NULL,
  gen_fun = gen_t.test,
  ...
)

gen_t.test(
  n,
  d,
  n2_n1 = 1,
  r = NULL,
  type = c("two.sample", "one.sample", "paired"),
  means = NULL,
  sds = NULL,
  ...
)
```

Arguments

n	sample size per group, assumed equal across groups
d	Cohen's standardized effect size d
mu	population mean to test against
r	(optional) instead of specifying d specify a point-biserial correlation. Internally this is transformed into a suitable d value for the power computations
type	type of t-test to use; can be 'two.sample', 'one.sample', or 'paired'

n2_n1	allocation ratio reflecting the same size ratio. Default of 1 sets the groups to be the same size. Only applicable when type = 'two.sample'
two.tailed	logical; should a two-tailed or one-tailed test be used?
var.equal	logical; use the classical or Welch corrected t-test?
means	(optional) vector of means for each group. When specified the input d is ignored
sds	(optional) vector of SDs for each group. When specified the input d is ignored
gen_fun	function used to generate the required two-sample data. Object returned must be a data.frame with the columns "DV" and "group". Default uses gen_t.test to generate conditionally Gaussian distributed samples. User defined version of this function must include the argument ...
...	additional arguments to be passed to gen_fun. Not used unless a customized gen_fun is defined

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

gen_t.test

Examples

```
# sample size of 50 per group, "medium" effect size
p_t.test(n=50, d=0.5)

# point-biserial correlation effect size
p_t.test(n=50, r=.3)

# second group 2x as large as the first group
p_t.test(n=50, d=0.5, n2_n1 = 2)

# specify mean/SDs explicitly
p_t.test(n=50, means = c(0,1), sds = c(2,2))

# paired and one-sample tests
p_t.test(n=50, d=0.5, type = 'paired')
p_t.test(n=50, d=0.5, type = 'one.sample')


  # compare simulated results to pwr package

  pwr::pwr.t.test(d=0.2, n=60, sig.level=0.10,
             type="one.sample", alternative="two.sided")
  p_t.test(n=60, d=0.2, type = 'one.sample', two.tailed=TRUE) |>
```

```
        Spower(sig.level=.10)

  pwr::pwr.t.test(d=0.3, power=0.80, type="two.sample",
                  alternative="greater")
  p_t.test(n=NA, d=0.3, type='two.sample', two.tailed=FALSE) |>
        Spower(power=0.80, interval=c(10,200))



###### Custom data generation function

# Generate data such that:
#   - group 1 is from a negatively distribution (reversed X2(10)),
#   - group 2 is from a positively skewed distribution (X2(5))
#   - groups have equal variance, but differ by d = 0.5

args(gen_t.test)   ## can use these arguments as a basis, though must include ...

# arguments df1 and df2 added; unused arguments caught within ...
my.gen_fun <- function(n, d, df1, df2, ...){
   group1 <- -1 * rchisq(n, df=df1)
     group2 <- rchisq(n, df=df2)
     # scale groups first given moments of the chi-square distribution,
     #   then add std mean difference
     group1 <- ((group1 + df1) / sqrt(2*df1))
     group2 <- ((group2 - df2) / sqrt(2*df2)) + d
     dat <- data.frame(DV=c(group1, group2),
            group=gl(2, n, labels=c('G1', 'G2')))
     dat
}

# check the sample data properties
df <- my.gen_fun(n=10000, d=.5, df1=10, df2=5)
with(df, tapply(DV, group, mean))
with(df, tapply(DV, group, sd))

library(ggplot2)
ggplot(df, aes(group, DV, fill=group)) + geom_violin()

p_t.test(n=100, d=0.5, gen_fun=my.gen_fun, df1=10, df2=5)



  # power given Gaussian distributions
  p_t.test(n=100, d=0.5) |> Spower(replications=30000)

  # estimate power given the customized data generating function
  p_t.test(n=100, d=0.5, gen_fun=my.gen_fun, df1=10, df2=5) |>
    Spower(replications=30000)

  # evaluate Type I error rate to see if liberal/conservative given
  # assumption violations (should be close to alpha/sig.level)
```

```
p_t.test(n=100, d=0, gen_fun=my.gen_fun, df1=10, df2=5) |>
  Spower(replications=30000)
```

`p_var.test` *p-value from variance test simulation*

Description

Generates one or or more sets of continuous data group-level data to perform a variance test, and return a p-value. When two-samples are investigated the `var.test` function will be used, otherwise functions from the `EnvStats` package will be used.

Usage

```
p_var.test(
  n,
  vars,
  n.ratios = rep(1, length(vars)),
  sigma2 = 1,
  two.tailed = TRUE,
  test = "Levene",
  correct = TRUE,
  gen_fun = gen_var.test,
  ...
)

gen_var.test(n, vars, n.ratios = rep(1, length(vars)), ...)
```

Arguments

`n`	sample size per group, assumed equal across groups
`vars`	a vector of variances to use for each group; length of 1 for one-sample tests
`n.ratios`	allocation ratios reflecting the sample size ratios. Default of 1 sets the groups to be the same size (n * n.ratio)
`sigma2`	population variance to test against in one-sample test
`two.tailed`	logical; should a two-tailed or one-tailed test be used?
`test`	type of test to use in multi-sample applications. Can be either `'Levene'` (default), `'Bartlett'`, or `'Fligner'`
`correct`	logical; use correction when `test = 'Bartlett'`?
`gen_fun`	function used to generate the required discrete data. Object returned must be a `matrix` with k rows and k columns of counts. Default uses `gen_var.test`. User defined version of this function must include the argument `...`
`...`	additional arguments to be passed to `gen_fun`. Not used unless a customized `gen_fun` is defined

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

gen_var.test

Examples

```
# one sample
p_var.test(100, vars=10, sigma2=9)

# three sample
p_var.test(100, vars=c(10, 9, 11))
p_var.test(100, vars=c(10, 9, 11), test = 'Fligner')
p_var.test(100, vars=c(10, 9, 11), test = 'Bartlett')


  # power to detect three-group variance differences
  p_var.test(n=100, vars=c(10,9,11)) |> Spower()

  # sample size per group to achieve 80% power
  p_var.test(n=NA, vars=c(10,9,11)) |>
         Spower(power=.80, interval=c(100, 1000))
```

---

p_wilcox.test *p-value from Wilcox test simulation*

Description

Simulates data given one or two parent distributions and returns a p-value. Can also be used for power analyses related to sign tests.

Usage

```
p_wilcox.test(
  n,
  d,
  n2_n1 = 1,
  mu = 0,
  type = c("two.sample", "one.sample", "paired"),
  exact = NULL,
```

```
  correct = TRUE,
  two.tailed = TRUE,
  parent1 = function(n, d) rnorm(n, d, 1),
  parent2 = function(n, d) rnorm(n, 0, 1)
)
```

Arguments

n	sample size per group
d	effect size passed to parent functions
n2_n1	sample size ratio
mu	parameter used to form the null hypothesis
type	type of analysis to use (two-sample, one-sample, or paired)
exact	a logical indicating whether an exact p-value should be computed
correct	a logical indicating whether to apply continuity correction in the normal approximation for the p-value
two.tailed	logical; use two-tailed test?
parent1	data generation function for first group. Ideally should have SDs = 1 so that d reflects a standardized difference
parent2	same as parent1, but for the second group

Value

a single p-value

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

Examples

```
# with normal distributions defaults d is standardized
p_wilcox.test(100, .5)
p_wilcox.test(100, .5, type = 'paired')
p_wilcox.test(100, .5, type = 'one.sample')

# using chi-squared distributions (standardizing to 0-1)
p_wilcox.test(100, .5, type = 'one.sample',
   parent1 = function(n, d) rchisq(n, df=10) - 10 + d)
p_wilcox.test(100, .5,
   parent1 = function(n, d) (rchisq(n, df=10) - 10)/sqrt(20) + d,
   parent2 = function(n, d) (rchisq(n, df=10) - 10)/sqrt(20))
```

Spower *Simulation-based Power Analyses*

Description

General purpose function that provides power-focused estimates for a priori, prospective/post-hoc, compromise, sensitivity, and criterion power analysis. Function provides a general wrapper to the SimDesign package's runSimulation and SimSolve functions. As such, parallel processing is automatically supported, along with progress bars, confidence/prediction intervals for the results estimates, safety checks, and more.

The function SpowerBatch, on the other hand, can be used to run Spower across different simulation combinations, returning a list of results instead. Can also be used as a pre-computing step before using SpowerCurve, and shares the same syntax specification (see SpowerCurve for further examples).

SpowerCurve draws power curves that either a) estimate the power given a set of varying conditions or b) solves a set of root conditions given fixed values of power. Confidence/prediction intervals are included in the output to reflect the estimate uncertainties, though note that fewer replications/iterations are used compared to Spower as the goal is visualization of competing variable inputs rather than precision of a given input.

Usage

```
Spower(
  ...,
  power = NA,
  sig.level = 0.05,
  interval,
  beta_alpha,
  sig.direction = "below",
  replications = 10000,
  integer,
  parallel = FALSE,
  cl = NULL,
  packages = NULL,
  ncores = parallelly::availableCores(omit = 1L),
  predCI = 0.95,
  predCI.tol = 0.01,
  verbose = TRUE,
  check.interval = FALSE,
  maxiter = 150,
  wait.time = NULL,
  lastSpower = NULL,
  select = NULL,
  control = list()
)
```

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

## S3 method for class 'Spower'
as.data.frame(x, ...)

SpowerBatch(
  ...,
  interval = NULL,
  power = NA,
  sig.level = 0.05,
  beta_alpha = NULL,
  sig.direction = "below",
  replications = 10000,
  integer,
  parallel = FALSE,
  cl = NULL,
  ncores = parallelly::availableCores(omit = 1L),
  predCI = 0.95,
  predCI.tol = 0.01,
  verbose = TRUE,
  check.interval = FALSE,
  maxiter = 150,
  wait.time = NULL,
  select = NULL,
  control = list()
)

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

## S3 method for class 'SpowerBatch'
as.data.frame(x, ...)

SpowerCurve(
  ...,
  interval = NULL,
  power = NA,
  sig.level = 0.05,
  sig.direction = "below",
  replications = 2500,
  integer,
  plotCI = TRUE,
  plotly = TRUE,
  parallel = FALSE,
  cl = NULL,
  ncores = parallelly::availableCores(omit = 1L),
  predCI = 0.95,
```

```
  predCI.tol = 0.01,
  verbose = TRUE,
  check.interval = FALSE,
  maxiter = 50,
  wait.time = NULL,
  select = NULL,
  batch = NULL,
  control = list()
)
```

Arguments

`...`	expression to use in the simulation that returns a `numeric` vector containing either the p-value (under the null hypothesis), the probability of the alternative hypothesis in the Bayesian setting, where the first numeric value in this vector is treated as the focus for all analyses other than prospective/post-hoc power. This corresponds to the `alpha` value used to flag samples as 'significant' when evaluating the null hypothesis (via p-values; $P(D\|H_0)$), where any returned p-value less that `sig.level` indicates significance. However, if `sig.direction = 'above'` then only values above `sig.level` are flagged as significant, which is useful in Bayesian posterior probability contexts that focus on the alternative hypothesis, $P(H_1\|D)$. Alternatively, a `logical` vector can be returned (e.g., when using confidence intervals (CIs) or evaluating regions of practical equivalence (ROPEs)), where the average of these TRUE/FALSE vector corresponds to the empirical power. For `SpowerCurve` and `SpowerBatch`, first expression input must be identical to `...` in `Spower`, while the remaining named inputs must match the arguments to this expression to indicate which variables should be modified in the resulting power curves. Providing `NA` values is also supported to solve the missing component. Note that only the first three named arguments in `SpowerCurve` will be plotted using the x-y, colour, and facet wrap aesthetics, respectively. However, if necessary the data can be extracted for further visualizations via `ggplot_build` to provide more customized control
`power`	power level to use. If set to `NA` (default) then the empirical power will be estimated given the fixed `...` inputs (e.g., for prospective/post-hoc power analysis). For `SpowerCurve` and `SpowerBatch` this can be a vector
`sig.level`	alpha level to use (default is `.05`). If set to `NA` then the value will be estimated given the fixed `conditions` input (e.g., for criterion power analysis). Only used when the value returned from the experiment is a `numeric` (e.g., a p-value, or a posterior probability; see `sig.direction`). If the return of the supplied experiment is a `logical` then this argument will be entirely ignored. As such, arguments such as `conf.level` should be included in the simulation experiment definition itself to indicate the explicit inferential criteria, and so that this argument can be manipulated should the need arise.
`interval`	required search interval to use when `SimSolve` is called to perform stochastic root solving. Note that for compromise analyses, where the `sig.level` is set to `NA`, if not set explicitly then the interval will default to `c(0,1)`

beta_alpha
: (optional) ratio to use in compromise analyses corresponding to the Type II errors (beta) over the Type I error (alpha). Ratios greater than $q = \beta/\alpha = 1$ indicate that Type I errors are worse than Type II, while ratios less than one the opposite. A ratio equal to 1 gives an equal trade-off between Type I and Type II errors

sig.direction
: a character vector that is either `'below'` (default) or `'above'` to indicate which direction relative to `sig.level` is considered significant. This is useful, for instance, when forming cutoffs for Bayesian posterior probabilities organized to show support for the hypothesis of interest ($P(H_1|D)$). As an example, setting `sig.level = .95` with `sig.direction = 'above'` flags a sample as 'significant' whenever the posterior probability is greater than .95.

replications
: number of replications to use when `runSimulation` is required. Default is 10000, though set to 2500 for `SpowerCurve`

integer
: a logical value indicating whether the search iterations use integers or doubles.

  If missing, automatically set to `FALSE` if `interval` contains non-integer numbers or the range is less than 5, as well as when `sig.level = NA`

parallel
: for parallel computing for slower simulation experiments (see `runSimulation` for details).

cl
: see `runSimulation`

packages
: see `runSimulation`

ncores
: see `runSimulation`

predCI
: predicting confidence interval level (see `SimSolve`)

predCI.tol
: predicting confidence interval consistency tolerance for stochastic root solver convergence (see `SimSolve`). Default converges when the power rate CI is consistently within `.01/2` of the target power

verbose
: logical; should information be printed to the console?

check.interval
: logical; check the interval range validity (see `SimSolve`). Disabled by default

maxiter
: maximum number of stochastic root-solving iterations. Default is 150, though set to 50 for `SpowerCurve`

wait.time
: (optional) argument to indicate the time to wait (specified in minutes if supplied as a numeric vector). See `SimSolve` for details and See `timeFormater` for further specifications

lastSpower
: a previously returned `Spower` object to be updated. Use this if you want to continue where an estimate left off but wish to increase the precision (e.g., by adding more replications, or by letting the stochastic root solver continue searching).

  Note that if the object was not stored use `getLastSpower` to obtain the last estimated power object

select
: a character vector indicating which elements to extract from the provided stimulation experiment function. By default, all elements from the provided function will be used, however if the provided function contains information not relevant to the power computations (e.g., parameter estimates, standard errors, etc) then these should be ignored. To extract the complete results post-analysis use `SimResults` to allow manual summarizing of the stored results (applicable only with prospective/post-hoc power)

control	a list of control parameters to pass to runSimulation or SimSolve
x	object of class 'Spower'. If SpowerBatch were used the this will be a list
plotCI	logical; include confidence/prediction intervals in plots?
plotly	logical; draw the graphic into the interactive plotly interface? If FALSE the ggplot2 object will be returned instead
batch	if SpowerBatch were previously used to perform the computations then this information can be provided to this batch argument to avoid recomputing

Details

Five types of power analysis flavors can be performed with Spower, which are triggered based on which supplied input is set to missing (NA):

A Priori Solve for a missing sample size component (e.g., n) to achieve a specific target power rate

Prospective and Post-hoc Estimate the power rate given a set of fixed conditions. If estimates of effect sizes and other empirical characteristics (e.g., observed sample size) are supplied this results in observed/retrospective power (not recommended), while if only sample size is included as the observed quantity, but the effect sizes are treated as unknown, then this results in post-hoc power (Cohen, 1988)

Sensitivity Solve a missing effect size value as a function of the other supplied constant components

Criterion Solve the error rate (argument sig.level) as a function of the other supplied constant components

Compromise Solve a Type I/Type II error trade-off ratio as a function of the other supplied constant components and the target ratio $q = \beta/\alpha$ (argument beta_alpha)

To understand how the package is structured, the first expression in the ... argument, which contains the simulation experiment definition for a single sample, is passed to either SimSolve or runSimulation depending on which element (including the power and sig.level arguments) is set to NA. For instance, Spower(p_t.test(n=50, d=.5)) will perform a prospective/post-hoc power evaluation since power = NA by default, while Spower(p_t.test(n=NA, d=.5), power = .80) will perform an a priori power analysis to solve the missing n argument.

For expected power computations, the arguments to the simulation experiment arguments can be specified as a function to reflect the prior uncertainty. For instance, if d_prior <- function() rnorm(1, mean=.5, sd=1/8) then Spower(p_t.test(n=50, d=d_prior())) will compute the expected power over the prior sampling distribution for d

Value

an invisible tibble/data.frame-type object of class 'Spower' containing the power results from the simulation experiment

a ggplot2 object automatically rendered with plotly for interactivity

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

See Also

update, SpowerCurve, getLastSpower, is.CI_within, is.outside_CI

Spower, SpowerBatch

Examples

```
############################
# Independent samples t-test
############################

# Internally defined p_t.test function
args(p_t.test)    # missing arguments required
# help(p_t.test)  # additional information

# p_* functions generate data and return single p-value
p_t.test(n=50, d=.5)
p_t.test(n=50, d=.5)

# test that it works
Spower(p_t.test(n = 50, d = .5), replications=10)

# also behaves naturally with a pipe
p_t.test(n = 50, d = .5) |> Spower(replications=10)



# Estimate power given fixed inputs (prospective power analysis)
out <- Spower(p_t.test(n = 50, d = .5))
summary(out)   # extra information
as.data.frame(out)  # coerced to data.frame

# increase precision (not run)
# p_t.test(n = 50, d = .5) |> Spower(replications=30000)

# alternatively, increase precision from previous object.
#   Here we add 20000 more replications on top of the previous 10000
p_t.test(n = 50, d = .5) |>
  Spower(replications=20000, lastSpower=out) -> out2
out2$REPLICATIONS  # total of 30000 replications for estimate

# previous analysis not stored to object, but can be retrieved
out <- getLastSpower()
out   # as though it were stored from Spower()

# Same as above, but executed with multiple cores (not run)
p_t.test(n = 50, d = .5) |>
   Spower(replications=30000, parallel=TRUE, ncores=2)

# Solve N to get .80 power (a priori power analysis)
p_t.test(n = NA, d = .5) |>
  Spower(power=.8, interval=c(2,500)) -> out
summary(out)  # extra information
```

```
plot(out)
plot(out, type = 'history')

# total sample size required
ceiling(out$n) * 2

# same as above, but in parallel with 2 cores
out.par <- p_t.test(n = NA, d = .5) |>
  Spower(power=.8, interval=c(2,500), parallel=TRUE, ncores=2)
summary(out.par)

# similar information from pwr package
(pwr <- pwr::pwr.t.test(d=.5, power=.80))
ceiling(pwr$n) * 2

# If greater precision is required and the user has a specific amount of time
# they are willing to wait (e.g., 5 minutes) then wait.time can be used. Below
# estimates root after searching for 1 minute, and run in parallel
#  with 2 cores (not run)
p_t.test(n = NA, d = .5) |>
  Spower(power=.8, interval=c(2,500), wait.time='1', parallel=TRUE, ncores=2)

# Similiar to above for precision improvements, however letting
#  the root solver continue searching from an early search history.
#  Usually a good idea to increase the maxiter and lower the predCI.tol
p_t.test(n = NA, d = .5) |>
  Spower(power=.8, interval=c(2,500), lastSpower=out,
        maxiter=200, predCI.tol=.008) #starts at last iteration in "out"

# Solve d to get .80 power (sensitivity power analysis)
p_t.test(n = 50, d = NA) |> Spower(power=.8, interval=c(.1, 2))
pwr::pwr.t.test(n=50, power=.80) # compare

# Solve alpha that would give power of .80 (criterion power analysis)
#    interval not required (set to interval = c(0, 1))
p_t.test(n = 50, d = .5) |> Spower(power=.80, sig.level=NA)

# Solve beta/alpha ratio to specific error trade-off constant
#   (compromise power analysis)
out <- p_t.test(n = 50, d = .5) |> Spower(beta_alpha = 2)
with(out, (1-power)/sig.level)   # solved ratio

# update beta_alpha criteria without re-simulating
(out2 <- update(out, beta_alpha=4))
with(out2, (1-power)/sig.level)   # solved ratio

###############
# Power Curves
###############

# SpowerCurve() has similar input, though requires varying argument
p_t.test(d=.5) |> SpowerCurve(n=c(30, 60, 90))
```

```
# solve n given power and plot
p_t.test(n=NA, d=.5) |> SpowerCurve(power=c(.2, .5, .8), interval=c(2,500))

# multiple varying components
p_t.test() |> SpowerCurve(n=c(30,60,90), d=c(.2, .5, .8))

################
# Expected Power
################

# Expected power computed by including effect size uncertainty.
# For instance, belief is that the true d is somewhere around d ~ N(.5, 1/8)
dprior <- function(x, mean=.5, sd=1/8) dnorm(x, mean=mean, sd=sd)
curve(dprior, -1, 2, main=expression(d %~% N(0.5, 1/8)),
      xlab='d', ylab='density')

# For Spower, define prior sampler for specific parameter(s)
d_prior <- function() rnorm(1, mean=.5, sd=1/8)
d_prior(); d_prior(); d_prior()

# Replace d constant with d_prior to compute expected power
p_t.test(n = 50, d = d_prior()) |> Spower()

# A priori power analysis using expected power
p_t.test(n = NA, d = d_prior()) |>
  Spower(power=.8, interval=c(2,500))
pwr::pwr.t.test(d=.5, power=.80) # expected power result higher than fixed d


###############
# Customization
###############

#   Make edits to the function for customization
if(interactive()){
    p_my_t.test <- edit(p_t.test)
    args(p_my_t.test)
    body(p_my_t.test)
}

# Alternatively, define a custom function (potentially based on the template)
p_my_t.test <- function(n, d, var.equal=FALSE, n2_n1=1, df=10){

    # Welch power analysis with asymmetric distributions
    # group2 as large as group1 by default

    # degree of skewness controlled via chi-squared distribution's df
    group1 <- rchisq(n, df=df)
    group1 <-  (group1 - df) / sqrt(2*df)   # Adjusted mean to 0, sd = 1
    group2 <- rnorm(n*n2_n1, mean=d)
    dat <- data.frame(group = factor(rep(c('G1', 'G2'),
                                     times = c(n, n*n2_n1))),
       DV = c(group1, group2))
```

```
    obj <- t.test(DV ~ group, dat, var.equal=var.equal)
    p <- obj$p.value
    p
}

# Solve N to get .80 power (a priori power analysis), using defaults
p_my_t.test(n = NA, d = .5, n2_n1=2) |>
  Spower(power=.8, interval=c(2,500)) -> out

# total sample size required
with(out, ceiling(n) + ceiling(n * 2))

# Solve N to get .80 power (a priori power analysis), assuming
#   equal variances, group2 2x as large as group1, large skewness
p_my_t.test(n = NA, d=.5, var.equal=TRUE, n2_n1=2, df=3) |>
  Spower(power=.8, interval=c(30,100)) -> out2

# total sample size required
with(out2, ceiling(n) + ceiling(n * 2))

# prospective power, can be used to extract the adjacent information
p_my_t.test(n = 100, d = .5) |> Spower() -> post

##############################
# Using CIs instead of p-values
##############################

# CI test returning TRUE if psi0 is outside the 95% CI
ci_ind.t.test <- function(n, d, psi0=0, conf.level=.95){
  g1 <- rnorm(n)
  g2 <- rnorm(n, mean=d)
  CI <- t.test(g2, g1, var.equal=TRUE,conf.level=conf.level)$conf.int
  is.outside_CI(psi0, CI)
}

# returns logical
ci_ind.t.test(n=100, d=.2)
ci_ind.t.test(n=100, d=.2)

# simulated prospective power
ci_ind.t.test(n=100, d=.2) |> Spower()

# compare to pwr package
pwr::pwr.t.test(n=100, d=.2)

############################
# Equivalence test power using CIs
#
# H0: population d is outside interval [LB, UB] (not tolerably equivalent)
# H1: population d is within interval [LB, UB]  (tolerably equivalent)

# CI test returning TRUE if CI is within tolerable equivalence range (tol)
ci_equiv.t.test <- function(n, d, tol, conf.level=.95){
```

```
  g1 <- rnorm(n)
  g2 <- rnorm(n, mean=d)
  CI <- t.test(g2, g1, var.equal=TRUE,conf.level=conf.level)$conf.int
  is.CI_within(CI, tol)
}

# evaluate if CI is within tolerable interval (tol)
ci_equiv.t.test(n=1000, d=.2, tol=c(.1, .3))

# simulated prospective power
ci_equiv.t.test(n=1000, d=.2, tol=c(.1, .3)) |> Spower()

# higher power with larger N (more precision) or wider tol interval
ci_equiv.t.test(n=2000, d=.2, tol=c(.1, .3)) |> Spower()
ci_equiv.t.test(n=1000, d=.2, tol=c(.1, .5)) |> Spower()

####
# superiority test (one-tailed)
# H0: population d is less than LB    (not superior)
# H1: population d is greater than LB (superior)

# set upper bound to Inf as it's not relevant, and reduce conf.level
#   to reflect one-tailed test
ci_equiv.t.test(n=1000, d=.2, tol=c(.1, Inf), conf.level=.90) |>
  Spower()

# higher LB means greater requirement for defining superiority (less power)
ci_equiv.t.test(n=1000, d=.2, tol=c(.15, Inf), conf.level=.90) |>
  Spower()



################################################
# SpowerBatch() examples
################################################

## Not run:

# estimate power given varying sample sizes
p_t.test(d=0.2) |>
  SpowerBatch(n=c(30, 90, 270, 550), replications=1000) -> nbatch
nbatch

# can be stacked to view the output as data.frame
as.data.frame(nbatch)

# plot with SpowerCurve()
SpowerCurve(batch=nbatch)

# equivalent, but re-runs the computations
p_t.test(d=0.2) |> SpowerCurve(n=c(30, 90, 270, 550), replications=1000)

# estimate power given varying sample sizes and effect size
```

```
p_t.test() |> SpowerBatch(n=c(30, 90, 270, 550),
                          d=c(.2, .5, .8), replications=1000) -> ndbatch
ndbatch

# plot with SpowerCurve()
SpowerCurve(batch=ndbatch)


## End(Not run)




##############################################
# SpowerCurve() examples
##############################################

# estimate power given varying sample sizes
gg <- p_t.test(d=0.2) |> SpowerCurve(n=c(30, 90, 270, 550))

# Output is a ggplot2 (rendered with plotly by default); hence, can be modified
library(ggplot2)
gg + geom_text(aes(label=power), size=5, colour='red', nudge_y=.05) +
  ylab(expression(1-beta)) + theme_grey()

# Increase precision by using 10000 replications. Parallel computations
#   generally recommended in this case to save time
p_t.test(d=0.2) |> SpowerCurve(n=c(30, 90, 270, 550), replications=10000)

# estimate sample sizes given varying power
p_t.test(n=NA, d=0.2) |>
  SpowerCurve(power=c(.2, .4, .6, .8), interval=c(10, 1000))

# get information from last printed graphic instead of saving
gg <- last_plot()
gg + coord_flip() # flip coordinates to put power on y-axis

# estimate power varying d
p_t.test(n=50) |> SpowerCurve(d=seq(.1, 1, by=.2))

# estimate d varying power
p_t.test(n=50, d=NA) |>
  SpowerCurve(power=c(.2, .4, .6, .8), interval=c(.01, 1))


#####

# vary two inputs instead of one (second input uses colour aesthetic)
p_t.test() |> SpowerCurve(n=c(30, 90, 270, 550),
                          d=c(.2, .5, .8))

# extract data for alternative presentations
build <- ggplot_build(last_plot())
```

```
build

df <- build$plot$data
head(df)
ggplot(df, aes(n, power, linetype=d)) + geom_line()

# vary three arguments (third uses facet_wrap ... any more than that and
#   you're on your own!)
p_t.test() |> SpowerCurve(n=c(30, 90, 270, 550),
                          d=c(.2, .5, .8),
                          var.equal=c(FALSE, TRUE))

########################################

# If objects were precomputed using SpowerBatch() then
#  these can be plotted instead
p_t.test(d=0.2) |>
  SpowerBatch(n=c(30, 90, 270, 550), replications=1000) -> nbatch
nbatch
as.data.frame(nbatch)

# plot the results, but avoid further computations
SpowerCurve(batch=nbatch)
```

`update.Spower` *Update compromise or prospective/post-hoc power analysis without re-simulating*

Description

When a power or compromise analysis was performed in `Spower` this function can be used to update the compromise or power criteria without the need for re-simulating the experiment. For compromise analyses a `beta_alpha` criteria must be supplied, while for prospective/post-hoc power analyses the `sig.level` must be supplied.

Usage

```
## S3 method for class 'Spower'
update(object, sig.level = 0.05, beta_alpha = NULL, predCI = 0.95, ...)
```

Arguments

`object`	object returned from `Spower` where `power` was estimated or the `bete_alpha` criteria were supplied
`sig.level`	Type I error rate (alpha)
`beta_alpha`	Type II/Type I error ratio

predCI	confidence interval precision (see Spower for similar input)
...	arguments to be passed

Value

object of class Spower with updated information

Author(s)

Phil Chalmers <rphilip.chalmers@gmail.com>

Examples

```
########
## Prospective power analysis update

# Estimate power using sig.level = .05 (default)
out <- p_t.test(n = 50, d = .5) |> Spower()

# update power estimate given sig.level=.01 and .20
update(out, sig.level=.01)
update(out, sig.level=.20)


########
## Compromise analysis update

# Solve beta/alpha ratio to specific error trade-off constant
out <- p_t.test(n = 50, d = .5) |> Spower(beta_alpha = 2)

# update beta_alpha criteria without re-simulating
update(out, beta_alpha=4)

# also works if compromise not initially run but prospective/post-hoc power was
out <- p_t.test(n = 50, d = .5) |> Spower()
update(out, beta_alpha=4)
```

Index