Package ‘safestats’

July 23, 2025

Type Package

Title Safe Anytime-Valid Inference

Version 0.8.7

Maintainer Alexander Ly <`a.ly@jasp-stats.org`>

Description Functions to design and apply tests that are anytime valid. The functions can be used to design hypothesis tests in the prospective/randomised control trial setting or in the observational/retrospective setting. The resulting tests remain valid under both optional stopping and optional continuation. The current version includes safe t-tests and safe tests of two proportions. For details on the theory of safe tests, see Grunwald, de Heide and Koolen (2019) ``Safe Testing'' <doi:10.48550/arXiv.1906.07801>, for details on safe logrank tests see ter Schure, Perez-Ortiz, Ly and Grunwald (2020) ``The Safe Logrank Test: Error Control under Continuous Monitoring with Unlimited Horizon'' <doi:10.48550/arXiv.2011.06931v3> and Turner, Ly and Grunwald (2021) ``Safe Tests and Always-Valid Confidence Intervals for contingency tables and beyond'' <doi:10.48550/arXiv.2106.02693> for details on safe contingency table tests.

License LGPL (>= 3)

Encoding UTF-8

Depends R (>= 3.6)

Imports stats (>= 3.6), hypergeo (>= 1.2-13), survival (>= 3.2-13), BiasedUrn (>= 1.07), boot (>= 1.3-28), dplyr (>= 1.0.6), purrr (>= 0.3.5), rlang (>= 1.0.6)

Suggests testthat (>= 3.0.0), knitr, rmarkdown, graphics

VignetteBuilder knitr

NeedsCompilation no

RoxygenNote 7.2.2

Language en-GB

Config/testthat/edition 3

Author Rosanne Turner [aut],
Alexander Ly [cre, aut],
Muriel Felipe Perez-Ortiz [ctb],
Judith ter Schure [ctb],
Peter Grunwald [ctb]

Repository CRAN

Date/Publication 2022-11-24 14:40:02 UTC

Contents

`checkAndReturnsEsMinParameterSide`

Checks consistency between the sided of the hypothesis and the minimal clinically relevant effect size or safe test defining parameter. Throws an error if the one-sided hypothesis is incongruent with the

---

Description

Checks consistency between the sided of the hypothesis and the minimal clinically relevant effect size or safe test defining parameter. Throws an error if the one-sided hypothesis is incongruent with the

Usage

```
checkAndReturnsEsMinParameterSide(
  paramToCheck,
  alternative = c("twoSided", "greater", "less"),
 esMinName = c("noName", "meanDiffMin", "phiS", "deltaMin", "deltaS", "hrMin", "thetaS",
    "deltaTrue"),
  paramDomain = NULL
)
```

Arguments

`paramToCheck`	numeric. Either a named safe test defining parameter such as phiS, or thetaS, or a minimal clinically relevant effect size called with a non-null esMinName name
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`esMinName`	provides the name of the effect size. Either "meanDiffMin" for the z-test, "deltaMin" for the t-test, or "hrMin" for the logrank test
`paramDomain`	Domain of the paramToCheck, typically, positiveNumbers. Default `NULL`

Value

paramToCheck after checking, perhaps with a change in sign

`checkAndReturnsNPlan` *Check consistency between nPlan and the testType for one and two-sample z and t-tests*

Description

Check consistency between nPlan and the testType for one and two-sample z and t-tests

Usage

```
checkAndReturnsNPlan(
  nPlan,
  ratio = 1,
  testType = c("oneSample", "paired", "twoSample")
)
```

Arguments

`nPlan`	optional numeric vector of length at most 2. When provided, it is used to find the safe test defining parameter phiS. Note that if the purpose is to plan based on nPlan alone, then both meanDiffMin and beta should be set to NULL.
`ratio`	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.
`testType`	either one of "oneSample", "paired", "twoSample".

Value

nPlan a vector of sample sizes of length 1 or 2

---

checkDoubleArgumentsDesignObject

Helper function to check whether arguments are specified in a function at a higher level and already provided in the design object.

Description

Helper function to check whether arguments are specified in a function at a higher level and already provided in the design object.

Usage

```
checkDoubleArgumentsDesignObject(designObj, ...)
```

Arguments

`designObj`	an object of class "safeDesign".
`...`	arguments that need checking.

Value

Returns nothing only used for its side-effects to produces warnings if needed.

Examples

```
designObj <- designSafeZ(0.4)

checkDoubleArgumentsDesignObject(designObj, "alpha"=NULL, alternative=NULL)
# Throws a warning
checkDoubleArgumentsDesignObject(designObj, "alpha"=0.4, alternative="d")
```

---

computeBetaBatchSafeZ

Helper function: Computes the type II error based on the minimal clinically relevant effect size and sample size.

Description

Helper function: Computes the type II error based on the minimal clinically relevant effect size and sample size.

Usage

```
computeBetaBatchSafeZ(
  meanDiffMin,
  nPlan,
  alpha = 0.05,
  sigma = 1,
  kappa = sigma,
  alternative = c("twoSided", "greater", "less"),
  testType = c("oneSample", "paired", "twoSample"),
  parameter = NULL
)
```

Arguments

`meanDiffMin`	numeric that defines the minimal relevant mean difference, the smallest population mean that we would like to detect.
`nPlan`	optional numeric vector of length at most 2. When provided, it is used to find the safe test defining parameter phiS. Note that if the purpose is to plan based on nPlan alone, then both meanDiffMin and beta should be set to NULL.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`sigma`	numeric > 0 representing the assumed population standard deviation used for the test.
`kappa`	the true population standard deviation. Default kappa=sigma.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`testType`	either one of "oneSample", "paired", "twoSample".
`parameter`	optional test defining parameter. Default set to `NULL`.

Value

numeric that represents the type II error

`computeBetaSafeT` *Helper function: Computes the type II error of the safeTTest based on the minimal clinically relevant standardised mean difference and nPlan.*

Description

Helper function: Computes the type II error of the safeTTest based on the minimal clinically relevant standardised mean difference and nPlan.

Usage

```
computeBetaSafeT(
  deltaMin,
  nPlan,
  alpha = 0.05,
  alternative = c("twoSided", "greater", "less"),
  testType = c("oneSample", "paired", "twoSample"),
  seed = NULL,
  parameter = NULL,
  pb = TRUE,
  nSim = 1000L,
  nBoot = 1000L
)
```

Arguments

`deltaMin`	numeric that defines the minimal relevant standardised effect size, the smallest effect size that we would the experiment to be able to detect.
`nPlan`	vector of max length 2 representing the planned sample sizes.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent of n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`testType`	either one of "oneSample", "paired", "twoSample".
`seed`	integer, seed number.
`parameter`	optional test defining parameter. Default set to `NULL`.
`pb`	logical, if `TRUE`, then show progress bar.
`nSim`	integer > 0, the number of simulations needed to compute power or the number of samples paths for the safe z test under continuous monitoring.
`nBoot`	integer > 0 representing the number of bootstrap samples to assess the accuracy of approximation of the power, the number of samples for the safe z test under continuous monitoring, or for the computation of the logarithm of the implied target.

Value

a list which contains at least beta and an adapted bootObject of class `boot`.

Examples

```
computeBetaSafeT(deltaMin=0.7, 27, nSim=10)
```

computeBetaSafeZ *Helper function: Computes the type II error based on the minimal clinically relevant mean difference and nPlan*

Description

Helper function: Computes the type II error based on the minimal clinically relevant mean difference and nPlan

Usage

```
computeBetaSafeZ(
  meanDiffMin,
  nPlan,
  alpha = 0.05,
  alternative = c("twoSided", "greater", "less"),
  sigma = 1,
  kappa = sigma,
  testType = c("oneSample", "paired", "twoSample"),
  parameter = NULL,
  pb = TRUE,
  nSim = 1000L,
  nBoot = 1000L
)
```

Arguments

`meanDiffMin`	numeric that defines the minimal relevant mean difference, the smallest population mean that we would like to detect.
`nPlan`	optional numeric vector of length at most 2. When provided, it is used to find the safe test defining parameter phiS. Note that if the purpose is to plan based on nPlan alone, then both meanDiffMin and beta should be set to NULL.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`sigma`	numeric > 0 representing the assumed population standard deviation used for the test.
`kappa`	the true population standard deviation. Default kappa=sigma.
`testType`	either one of "oneSample", "paired", "twoSample".
`parameter`	optional test defining parameter. Default set to `NULL`.
`pb`	logical, if `TRUE`, then show progress bar.
`nSim`	integer > 0, the number of simulations needed to compute power or the number of samples paths for the safe z test under continuous monitoring.

`nBoot`	integer > 0 representing the number of bootstrap samples to assess the accuracy of approximation of the power, the number of samples for the safe z test under continuous monitoring, or for the computation of the logarithm of the implied target.

Value

a list which contains at least beta and an adapted bootObject of class `boot`.

Examples

```
computeBetaSafeZ(meanDiffMin=0.7, 20, nSim=10)
```

`computeBootObj` *Computes the bootObj for sequential sampling procedures regarding nPlan, beta, the implied target*

Description

Computes the bootObj for sequential sampling procedures regarding nPlan, beta, the implied target

Usage

```
computeBootObj(
  values,
  beta = NULL,
  nPlan = NULL,
  nBoot = 1000L,
  alpha = NULL,
  objType = c("nPlan", "nMean", "beta", "betaFromEValues", "logImpliedTarget",
    "expectedStopTime")
)
```

Arguments

`values`	numeric vector. If objType equals "nPlan" or "beta" then values should be stopping times, if objType equals "logImpliedTarget" then values should be eValues.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both "n" and "phiS". Note that 1-beta defines the power.
`nPlan`	integer > 0 representing the number of planned samples (for the first group).
`nBoot`	integer > 0 representing the number of bootstrap samples to assess the accuracy of approximation of the power, the planned sample size(s) of the safe test under continuous monitoring.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`objType`	character string either "nPlan", "nMean", "beta", "betaFromEValues", "expectedStopTime" or "logImpliedTarget".

Value

bootObj

Examples

```
computeBootObj(1:100, objType="nPlan", beta=0.3)
```

`computeConfidenceBoundForLogOddsTwoProportions`
Estimate an upper or lower bound for a safe confidence sequence on the logarithm of the odds ratio for two proportions.

Description

Estimate an upper or lower bound for a safe confidence sequence on the logarithm of the odds ratio for two proportions.

Usage

```
computeConfidenceBoundForLogOddsTwoProportions(
  ya,
  yb,
  safeDesign,
  bound = c("lower", "upper"),
  deltaStart,
  deltaStop,
  precision
)
```

Arguments

`ya`	positive observations/ events per data block in group a: a numeric with integer values between (and including) 0 and `na`, the number of observations in group a per block.
`yb`	positive observations/ events per data block in group b: a numeric with integer values between (and including) 0 and `nb`, the number of observations in group b per block.
`safeDesign`	a 'safeDesign' object obtained through `designSafeTwoProportions`
`bound`	type of bound to calculate; "lower" to get a lower bound on positive delta, "upper" to get an upper bound on negative delta.
`deltaStart`	starting value of the grid to search over for the bound on the confidence sequence (in practice: the interval). Numeric >0 when searching for a lower bound, numeric < 0 when searching for an upper bound.
`deltaStop`	end value of the grid to search over for the bound on the confidence sequence (in practice: the interval). Numeric >0 when searching for a lower bound, numeric < 0 when searching for an upper bound.
`precision`	precision of the grid between deltaStart and deltaStop.

Value

numeric: the established lower- or upper bound on the logarithm of the odds ratio between the groups

Examples

```
balancedSafeDesign <- designSafeTwoProportions(na = 1,
                                               nb = 1,
                                               nBlocksPlan = 10,
                                               alpha = 0.05)
#hypothesize OR < 1 (i.e., log OR < 0)
ya <- c(1,1,1,1,1,1,1,1,0,1)
yb <- c(0,0,0,0,1,0,0,0,0,0)
#one-sided CI for OR-, establish upper bound on log odds ratio
computeConfidenceBoundForLogOddsTwoProportions(ya = ya,
                                           yb = yb,
                                           safeDesign = balancedSafeDesign,
                                           bound = "upper",
                                           deltaStart = -0.01,
                                           deltaStop = -4,
                                           precision = 20)
```

`computeConfidenceBoundsForDifferenceTwoProportions`
Estimate Lower and Upper Bounds on the Confidence Sequence (Interval) for the Difference Divergence Measure for Two Proportions

---

Description

Estimate Lower and Upper Bounds on the Confidence Sequence (Interval) for the Difference Divergence Measure for Two Proportions

Usage

```
computeConfidenceBoundsForDifferenceTwoProportions(
  ya,
  yb,
  precision,
  safeDesign
)
```

Arguments

`ya`	positive observations/ events per data block in group a: a numeric with integer values between (and including) 0 and `na`, the number of observations in group a per block.

yb	positive observations/ events per data block in group b: a numeric with integer values between (and including) 0 and nb, the number of observations in group b per block.
precision	precision of the grid of differences to search over for the lower and upper bounds.
safeDesign	a 'safeDesign' object obtained through designSafeTwoProportions

Value

list with found lower and upper bound.

Examples

```
balancedSafeDesign <- designSafeTwoProportions(na = 1,
                                               nb = 1,
                                               nBlocksPlan = 10,
                                               alpha = 0.05)
ya <- c(1,1,1,1,1,1,1,1,0,1)
yb <- c(0,0,0,0,1,0,0,0,0,0)
computeConfidenceBoundsForDifferenceTwoProportions(ya = ya,
                                               yb = yb,
                                               precision = 20,
                                               safeDesign = balancedSafeDesign)
```

---

computeConfidenceIntervalT   *Helper function: Computes the safe confidence sequence for the mean in a t-test*

Description

Helper function: Computes the safe confidence sequence for the mean in a t-test

Usage

```
computeConfidenceIntervalT(
  meanObs,
  sdObs,
  nEff,
  nu,
  deltaS,
  ciValue = 0.95,
  g = NULL
)
```

Arguments

`meanObs`	numeric, the observed mean. For two sample tests this is difference of the means.
`sdObs`	numeric, the observed standard deviation. For a two-sample test this is the root of the pooled variance.
`nEff`	numeric > 0, the effective sample size. For one sample test this is just n.
`nu`	numeric > 0, the degrees of freedom.
`deltaS`	numeric > 0, the safe test defining parameter.
`ciValue`	numeric is the ciValue-level of the confidence sequence. Default ciValue=0.95.
`g`	numeric > 0, used as the variance of the normal prior on the population delta Default is `NULL` in which case g=delta^2.

Value

numeric vector that contains the upper and lower bound of the safe confidence sequence

Examples

```
computeConfidenceIntervalT(meanObs=0.3, sdObs=2, nEff=12, nu=11, deltaS=0.4)
```

`computeConfidenceIntervalZ`
Helper function: Computes the safe confidence sequence for a z-test

---

Description

Helper function: Computes the safe confidence sequence for a z-test

Usage

```
computeConfidenceIntervalZ(
  nEff,
  meanObs,
  phiS,
  sigma = 1,
  ciValue = 0.95,
  alternative = "twoSided",
  a = NULL,
  g = NULL
)
```

Arguments

`nEff`	numeric > 0, the effective sample size.
`meanObs`	numeric, the observed mean. For two sample tests this is difference of the means.
`phiS`	numeric > 0, the safe test defining parameter.
`sigma`	numeric > 0 representing the assumed population standard deviation used for the test.
`ciValue`	numeric is the ciValue-level of the confidence sequence. Default ciValue=0.95.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`a`	numeric, the centre of the normal prior on population mean (of the normal data). Default is `NULL`, which implies the default choice of setting the centre equal to the null hypothesis.
`g`	numeric > 0, used to define g sigma^2 as the variance of the normal prior on the population (of the normal data). Default is `NULL` in which case g=phiS^2/sigma^2.

Value

numeric vector that contains the upper and lower bound of the safe confidence sequence

Examples

```
computeConfidenceIntervalZ(nEff=15, meanObs=0.3, phiS=0.2)
```

---

computeEsMinSafeT *Helper function: Computes the minimal clinically relevant standardised mean difference for the safe t-test nPlan and beta.*

Description

Helper function: Computes the minimal clinically relevant standardised mean difference for the safe t-test nPlan and beta.

Usage

```
computeEsMinSafeT(
  nPlan,
  alpha = 0.05,
  beta = 0.2,
  alternative = c("twoSided", "greater", "less"),
  testType = c("oneSample", "paired", "twoSample"),
  lowN = 3,
  highN = 1e+06,
  ratio = 1
)
```

Arguments

nPlan	vector of max length 2 representing the planned sample sizes.
alpha	numeric in (0, 1) that specifies the tolerable type I error control –independent of n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
beta	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both the sample sizes and deltaS, which defines the test. Note that 1-beta defines the power.
alternative	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
testType	either one of "oneSample", "paired", "twoSample".
lowN	integer minimal sample size of the (first) sample when computing the power due to optional stopping. Default lowN is set 1.
highN	integer minimal sample size of the (first) sample when computing the power due to optional stopping. Default highN is set 1e6.
ratio	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.

Value

a list which contains at least nPlan and the phiS the parameter that defines the safe test

---

computeLogrankBetaFrom

Helper function: Computes the type II error under optional stopping based on the minimal clinically relevant hazard ratio and the maximum number of nEvents.

Description

Helper function: Computes the type II error under optional stopping based on the minimal clinically relevant hazard ratio and the maximum number of nEvents.

Usage

```
computeLogrankBetaFrom(
  hrMin,
  nEvents,
  m0 = 50000L,
  m1 = 50000L,
  alpha = 0.05,
  alternative = c("twoSided", "greater", "less"),
  nSim = 1000L,
  nBoot = 10000L,
  groupSizePerTimeFunction = returnOne,
  parameter = NULL,
  pb = TRUE
)
```

Arguments

hrMin	numeric that defines the minimal relevant hazard ratio, the smallest hazard ratio that we want to detect.
nEvents	numeric > 0, targetted number of events.
m0	Number of subjects in the control group 0/1 at the beginning of the trial, i.e., nPlan[1].
m1	Number of subjects in the treatment group 1/2 at the beginning of the trial, i.e., nPlan[2].
alpha	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
alternative	a character string specifying the alternative hypothesis, which must be one of "twoSided" (default),"greater" or "less". The alternative is pitted against the null hypothesis of equality of the survival distributions. More specifically, let lambda1 be the hazard rate of group 1 (i.e., placebo), and lambda2 the hazard ratio of group 2 (i.e., treatment), then the null hypothesis states that the hazard ratio theta = lambda2/lambda1 = 1. If alternative = "less", the null hypothesis is compared to theta < 1, thus, lambda2 < lambda1, that is, the hazard of group 2 (i.e., treatment) is less than that of group 1 (i.e., placebo), hence, the treatment is beneficial. If alternative = "greater", then the null hypothesis is compared to theta > 1, thus, lambda2 > lambda1, hence, harm.
nSim	integer > 0, the number of simulations needed to compute power or the number of events for the exact safe logrank test under continuous monitoring
nBoot	integer > 0 representing the number of bootstrap samples to assess the accuracy of the approximation of power or nEvents for the exact safe logrank test under continuous monitoring
groupSizePerTimeFunction	A function without parameters and integer output. This function provides the number of events at each time step. For instance, if `rpois(1, 7)` leads to a random number of events at each time step.
parameter	Numeric > 0, represents the safe tests defining thetaS. Default NULL so it's decided by the algorithm, typically, this equals hrMin, which corresponds to the GROW choice.
pb	logical, if `TRUE`, then show progress bar.

Value

a list which contains at least beta and an adapted bootObject of class `boot`.

Author(s)

Muriel Felipe Perez-Ortiz and Alexander Ly

Examples

```
computeLogrankBetaFrom(hrMin=0.7, 300, nSim=10)
```

`computeLogrankNEvents` *Helper function: Computes the planned sample size based on the minimal clinical relevant hazard ratio, alpha and beta under optional stopping.*

Description

Helper function: Computes the planned sample size based on the minimal clinical relevant hazard ratio, alpha and beta under optional stopping.

Usage

```
computeLogrankNEvents(
  hrMin,
  beta,
  m0 = 50000,
  m1 = 50000,
  alpha = 0.05,
  alternative = c("twoSided", "greater", "less"),
  nSim = 1000L,
  nBoot = 1000L,
  groupSizePerTimeFunction = returnOne,
  nMax = Inf,
  parameter = NULL,
  digits = getOption("digits"),
  pb = TRUE
)
```

Arguments

`hrMin`	numeric that defines the minimal relevant hazard ratio, the smallest hazard ratio that we want to detect.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both "n" and "phiS". Note that 1-beta defines the power.
`m0`	Number of subjects in the control group 0/1 at the beginning of the trial, i.e., nPlan[1].
`m1`	Number of subjects in the treatment group 1/2 at the beginning of the trial, i.e., nPlan[2].

`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`alternative`	a character string specifying the alternative hypothesis, which must be one of "twoSided" (default),"greater" or "less". The alternative is pitted against the null hypothesis of equality of the survival distributions. More specifically, let lambda1 be the hazard rate of group 1 (i.e., placebo), and lambda2 the hazard ratio of group 2 (i.e., treatment), then the null hypothesis states that the hazard ratio theta = lambda2/lambda1 = 1. If alternative = "less", the null hypothesis is compared to theta < 1, thus, lambda2 < lambda1, that is, the hazard of group 2 (i.e., treatment) is less than that of group 1 (i.e., placebo), hence, the treatment is beneficial. If alternative = "greater", then the null hypothesis is compared to theta > 1, thus, lambda2 > lambda1, hence, harm.
`nSim`	integer > 0, the number of simulations needed to compute power or the number of events for the exact safe logrank test under continuous monitoring
`nBoot`	integer > 0 representing the number of bootstrap samples to assess the accuracy of the approximation of power or nEvents for the exact safe logrank test under continuous monitoring
`groupSizePerTimeFunction`	A function without parameters and integer output. This function provides the number of events at each time step. For instance, if `rpois(1, 7)` leads to a random number of events at each time step.
`nMax`	An integer. Once nEvents hits nMax the experiment terminates, if it didn't stop due to threshold crossing crossing already. Default set to Inf.
`parameter`	Numeric > 0, represents the safe tests defining thetaS. Default NULL so it's decided by the algorithm, typically, this equals hrMin, which corresponds to the GROW choice.
`digits`	number of significant digits to be used.
`pb`	logical, if `TRUE`, then show progress bar.

Value

a list which contains at least nEvents and an adapted bootObject of class `boot`.

Author(s)

Muriel Felipe Perez-Ortiz and Alexander Ly

Examples

```
computeLogrankNEvents(0.7, 0.2, nSim=10)
```

computeLogrankZ — *Helper function to computes the logrank statistic for 'Surv' objects of type "right" and "counting" with the hypergeometric variance.*

Description

This function was created to complement `survdiff` from the 'survival' package, which is restricted to 'Surv' objects of type "right". Most likely `survdiff` is much faster

Usage

```
computeLogrankZ(
  survObj,
  group,
  computeZ = TRUE,
  computeExactE = FALSE,
  theta0 = 1,
  thetaS = NULL,
  ...
)
```

Arguments

`survObj`	a Surv object that is either of type
`group`	a grouping factor with 2 levels
`computeZ`	logical. If `TRUE` computes the logrank z-statistic. Default is `TRUE`.
`computeExactE`	logical. If `TRUE` computes one-sided exact logrank e-value. Default is `FALSE`.
`theta0`	numeric > 0 used only for the e-value, i.e., if computeExactE is `TRUE`. Default is 1.
`thetaS`	numeric > 0 used only for the e-value, i.e., if computeExactE is `TRUE`. Default is NULL.
`...`	further arguments to be passed to or from methods.

Value

Returns a list containing at least the following components:

nEvents the number of events.

z the observed logrank statistic.

oMinEVector vector of observed minus expected.

varVector vector of hypergeometric variances.

stopTimeVector vector at which the events occurred.

Examples

```
data <- generateSurvData(nP = 5,
                         nT = 5,
                         lambdaP = 0.03943723,
                         lambdaT = 0.5*0.03943723,
                         endTime = 40,
                         seed = 2006)

survObj <- survival::Surv(data$time, data$status)

survObj <- survival::Surv(data$time, data$status)

result <- computeLogrankZ(survObj, data$group)
result$z
sqrt(survival::survdiff(survObj~data$group)$chisq)
```

---

computeMinEsBatchSafeZ

Computes the smallest mean difference that is detectable with chance 1-beta, for the provided sample size

---

Description

Computes the smallest mean difference that is detectable with chance 1-beta, for the provided sample size

Usage

```
computeMinEsBatchSafeZ(
  nPlan,
  alpha = 0.05,
  beta = 0.2,
  sigma = 1,
  kappa = sigma,
  alternative = c("twoSided", "greater", "less"),
  testType = c("oneSample", "paired", "twoSample"),
  parameter = NULL,
  maxIter = 10
)
```

Arguments

`nPlan`	optional numeric vector of length at most 2. When provided, it is used to find the safe test defining parameter phiS. Note that if the purpose is to plan based on nPlan alone, then both meanDiffMin and beta should be set to NULL.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.

`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both "n" and "phiS". Note that 1-beta defines the power.
`sigma`	numeric > 0 representing the assumed population standard deviation used for the test.
`kappa`	the true population standard deviation. Default kappa=sigma.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`testType`	either one of "oneSample", "paired", "twoSample".
`parameter`	optional test defining parameter. Default set to `NULL`.
`maxIter`	maximum number of iterations in the optimisation process for two-sided designs

Value

numeric > 0 that represents the minimal detectable mean difference

---

`computeNEff` *Help function to compute the effective sample size based on a length 2 vector of samples*

Description

Help function to compute the effective sample size based on a length 2 vector of samples

Usage

```
computeNEff(n, testType = c("oneSample", "paired", "twoSample"), silent = TRUE)
```

Arguments

`n`	vector of length at most 2 representing the sample sizes of the first and second group
`testType`	either one of "oneSample", "paired", "twoSample".
`silent`	logical, if true, then turn off warnings

Value

a numeric that represents the effective sample size.

computeNPlanBatchSafeT

Helper function: Computes the planned sample size for the safe t-test based on the minimal clinically relevant standardised effect size, alpha and beta.

Description

Helper function: Computes the planned sample size for the safe t-test based on the minimal clinically relevant standardised effect size, alpha and beta.

Usage

```
computeNPlanBatchSafeT(
  deltaMin,
  alpha = 0.05,
  beta = 0.2,
  alternative = c("twoSided", "greater", "less"),
  testType = c("oneSample", "paired", "twoSample"),
  lowN = 3,
  highN = 1e+06,
  ratio = 1
)
```

Arguments

`deltaMin`	numeric that defines the minimal relevant standardised effect size, the smallest effect size that we would the experiment to be able to detect.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent of n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both the sample sizes and deltaS, which defines the test. Note that 1-beta defines the power.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`testType`	either one of "oneSample", "paired", "twoSample".
`lowN`	integer minimal sample size of the (first) sample when computing the power due to optional stopping. Default lowN is set 1.
`highN`	integer minimal sample size of the (first) sample when computing the power due to optional stopping. Default highN is set 1e6.
`ratio`	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.

Value

a list which contains at least nPlan and the phiS the parameter that defines the safe test

---

`computeNPlanBatchSafeZ`

Helper function: Computes the planned sample size based on the minimal clinical relevant mean difference, alpha and beta.

Description

Helper function: Computes the planned sample size based on the minimal clinical relevant mean difference, alpha and beta.

Usage

```
computeNPlanBatchSafeZ(
  meanDiffMin,
  alpha = 0.05,
  beta = 0.2,
  sigma = 1,
  kappa = sigma,
  alternative = c("twoSided", "greater", "less"),
  testType = c("oneSample", "paired", "twoSample"),
  tol = 1e-05,
  highN = 1e+06,
  ratio = 1,
  parameter = NULL,
  grow = TRUE
)
```

Arguments

`meanDiffMin`	numeric that defines the minimal relevant mean difference, the smallest population mean that we would like to detect.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both "n" and "phiS". Note that 1-beta defines the power.
`sigma`	numeric > 0 representing the assumed population standard deviation used for the test.
`kappa`	the true population standard deviation. Default kappa=sigma.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`testType`	either one of "oneSample", "paired", "twoSample".

`tol`	a number that defines the stepsizes between the lowParam and highParam.
`highN`	integer that defines the largest n of our search space for n. This might be the largest n that we are able to fund.
`ratio`	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.
`parameter`	optional test defining parameter. Default set to `NULL`.
`grow`	logical, default set to `TRUE` so the grow safe test is used in the design.

Value

a list which contains at least nPlan and the phiS, that is, the parameter that defines the safe test.

`computeNPlanSafeT` *Helper function: Computes the planned sample size of the safe t-test based on the minimal clinical relevant standardised mean difference.*

Description

Helper function: Computes the planned sample size of the safe t-test based on the minimal clinical relevant standardised mean difference.

Usage

```
computeNPlanSafeT(
  deltaMin,
  beta = 0.2,
  alpha = 0.05,
  alternative = c("twoSided", "less", "greater"),
  testType = c("oneSample", "paired", "twoSample"),
  lowN = 3,
  highN = 1e+06,
  ratio = 1,
  nSim = 1000L,
  nBoot = 1000L,
  parameter = NULL,
  pb = TRUE,
  nMax = 1e+06,
  seed = NULL
)
```

Arguments

`deltaMin`	numeric that defines the minimal relevant standardised effect size, the smallest effect size that we would the experiment to be able to detect.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both the sample sizes and deltaS, which defines the test. Note that 1-beta defines the power.

`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent of n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`testType`	either one of "oneSample", "paired", "twoSample".
`lowN`	integer minimal sample size of the (first) sample when computing the power due to optional stopping. Default lowN is set 1.
`highN`	integer minimal sample size of the (first) sample when computing the power due to optional stopping. Default highN is set 1e6.
`ratio`	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.
`nSim`	integer > 0, the number of simulations needed to compute power or the number of samples paths for the safe z test under continuous monitoring.
`nBoot`	integer > 0 representing the number of bootstrap samples to assess the accuracy of approximation of the power, the number of samples for the safe z test under continuous monitoring, or for the computation of the logarithm of the implied target.
`parameter`	optional test defining parameter. Default set to `NULL`.
`pb`	logical, if `TRUE`, then show progress bar.
`nMax`	integer > 0, maximum sample size of the (first) sample in each sample path.
`seed`	integer, seed number.

Value

a list which contains at least nPlan and an adapted bootObject of class `boot`.

Examples

```
computeNPlanSafeT(0.7, 0.2, nSim=10)
```

`computeNPlanSafeZ` *Helper function: Computes the planned sample size based on the minimal clinical relevant mean difference, alpha and beta*

Description

Helper function: Computes the planned sample size based on the minimal clinical relevant mean difference, alpha and beta

Usage

```
computeNPlanSafeZ(
  meanDiffMin,
  beta = 0.2,
  alpha = 0.05,
  alternative = c("twoSided", "less", "greater"),
  testType = c("oneSample", "paired", "twoSample"),
  sigma = 1,
  kappa = sigma,
  ratio = 1,
  nSim = 1000L,
  nBoot = 1000L,
  parameter = NULL,
  pb = TRUE,
  nMax = 1e+08
)
```

Arguments

`meanDiffMin`	numeric that defines the minimal relevant mean difference, the smallest population mean that we would like to detect.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both "n" and "phiS". Note that 1-beta defines the power.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`testType`	either one of "oneSample", "paired", "twoSample".
`sigma`	numeric > 0 representing the assumed population standard deviation used for the test.
`kappa`	the true population standard deviation. Default kappa=sigma.
`ratio`	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.
`nSim`	integer > 0, the number of simulations needed to compute power or the number of samples paths for the safe z test under continuous monitoring.
`nBoot`	integer > 0 representing the number of bootstrap samples to assess the accuracy of approximation of the power, the number of samples for the safe z test under continuous monitoring, or for the computation of the logarithm of the implied target.
`parameter`	optional test defining parameter. Default set to `NULL`.
`pb`	logical, if `TRUE`, then show progress bar.
`nMax`	integer > 0, maximum sample size of the (first) sample in each sample path.

Value

a list which contains at least nPlan and an adapted bootObject of class `boot`.

Examples

```
computeNPlanSafeZ(0.7, 0.2, nSim=10)
```

`computeStatsForLogrank`
Computes the sufficient statistics needed to compute 'logrankSingleZ'

Description

Computes the sufficient statistics needed to compute 'logrankSingleZ'

Usage

```
computeStatsForLogrank(
  survDataFrame,
  y0Index,
  y1Index,
  timeNow,
  timeBefore,
  survType = "right",
  ...
)
```

Arguments

`survDataFrame`	a 'Surv' object converted to a matrix, then to a data.frame
`y0Index`	vector of integers corresponding to the control group
`y1Index`	vector of integers corresponding to the treatment group
`timeNow`	numeric, current time
`timeBefore`	numeric, previous time
`survType`	character, either "right" or "counting" (left truncated, right censored)
`...`	further arguments to be passed to or from methods.

Value

Returns a list containing at least the following components:

obs0 number of observations in the control group.

obs1 number of observations in the treatment group.

y0 total number of participants in the control group.

y1 total number of participants in the treatment group.#'

Examples

```
data <- generateSurvData(nP = 5,
                         nT = 5,
                         lambdaP = 0.03943723,
                         lambdaT = 0.5*0.03943723,
                         endTime = 40,
                         seed = 2006)

survObj <- survival::Surv(data$time, data$status)

survDataFrame <- as.data.frame(as.matrix(survObj))
y0Index <- which(data$group=="P")
y1Index <- which(data$group=="T")

timeNow <- 4
timeBefore <- 0

computeStatsForLogrank(survDataFrame, y0Index, y1Index, timeNow, timeBefore)

timeNow <- 13
timeBefore <- 4

computeStatsForLogrank(survDataFrame, y0Index, y1Index, timeNow, timeBefore)
```

`defineTTestN` *Computes a Sequence of (Effective) Sample Sizes*

Description

Helper function that outputs the sample sizes, effective sample sizes and the degrees of freedom depending on the type of t-test. Also used for z-tests.

Usage

```
defineTTestN(
  lowN = 3,
  highN = 100,
  ratio = 1,
  testType = c("oneSample", "paired", "twoSample")
)
```

Arguments

`lowN`	integer minimal sample size of the (first) sample when computing the power due to optional stopping. Default lowN is set 1.
`highN`	integer minimal sample size of the (first) sample when computing the power due to optional stopping. Default highN is set 1e6.

`ratio`	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.
`testType`	either one of "oneSample", "paired", "twoSample".

Value

Returns the sample sizes and degrees of freedom.

`designFreqT` *Design a Frequentist T-Test*

Description

Computes the number of samples necessary to reach a tolerable type I and type II error for the frequentist t-test.

Usage

```
designFreqT(
  deltaMin,
  alpha = 0.05,
  beta = 0.2,
  alternative = c("twoSided", "greater", "less"),
  h0 = 0,
  testType = c("oneSample", "paired", "twoSample"),
  ...
)
```

Arguments

`deltaMin`	numeric that defines the minimal relevant standardised effect size, the smallest effect size that we would the experiment to be able to detect.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent of n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both the sample sizes and deltaS, which defines the test. Note that 1-beta defines the power.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`h0`	a number indicating the hypothesised true value of the mean under the null. For the moment h0=0.
`testType`	either one of "oneSample", "paired", "twoSample".
`...`	further arguments to be passed to or from methods, but mainly to perform do.calls.

Value

Returns an object of class ’freqTDesign’. An object of class ’freqTDesign’ is a list containing at least the following components:

nPlan the planned sample size(s).

esMin the minimal clinically relevant standardised effect size provided by the user.

alpha the tolerable type I error provided by the user.

beta the tolerable type II error provided by the user.

lowN the smallest n of the search space for n provided by the user.

highN the largest n of the search space for n provided by the user.

testType any of "oneSample", "paired", "twoSample" provided by the user.

alternative any of "twoSided", "greater", "less" provided by the user.

Examples

```
designFreqT(0.5)
```

`designFreqZ` *Design a Frequentist Z-Test*

Description

Computes the number of samples necessary to reach a tolerable type I and type II error for the frequentist z-test.

Usage

```
designFreqZ(
  meanDiffMin,
  alternative = c("twoSided", "greater", "less"),
  alpha = 0.05,
  beta = 0.2,
  testType = c("oneSample", "paired", "twoSample"),
  ratio = 1,
  sigma = 1,
  h0 = 0,
  kappa = sigma,
  lowN = 3L,
  highN = 100L,
  ...
)
```

Arguments

`meanDiffMin`	numeric that defines the minimal relevant mean difference, the smallest population mean that we would like to detect.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both "n" and "phiS". Note that 1-beta defines the power.
`testType`	either one of "oneSample", "paired", "twoSample".
`ratio`	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.
`sigma`	numeric > 0 representing the assumed population standard deviation used for the test.
`h0`	numeric, represents the null hypothesis, default h0=0.
`kappa`	the true population standard deviation. Default kappa=sigma.
`lowN`	integer that defines the smallest n of our search space for n.
`highN`	integer that defines the largest n of our search space for n. This might be the largest n that we are able to fund.
`...`	further arguments to be passed to or from methods.

Value

returns a 'freqZDesign' object.

Examples

```
freqDesign <- designFreqZ(meanDiffMin = 0.5, highN = 100)
freqDesign$nPlan
freqDesign2 <- designFreqZ(meanDiffMin = 0.2, lowN = 32, highN = 200)
freqDesign2$nPlan
```

`designPilotSafeT` *Designs a Safe T-Test Based on Planned Samples nPlan*

Description

Designs a safe experiment for a prespecified tolerable type I error based on planned sample size(s), which are fixed ahead of time. Outputs a list that includes the deltaS, i.e., the safe test defining parameter.

Usage

```
designPilotSafeT(
  nPlan = 50,
  alpha = 0.05,
  alternative = c("twoSided", "greater", "less"),
  h0 = 0,
  lowParam = 0.01,
  highParam = 1.2,
  tol = 0.01,
  inverseMethod = TRUE,
  logging = FALSE,
  paired = FALSE,
  maxIter = 10
)
```

Arguments

`nPlan`	the planned sample size(s).
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent of n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`h0`	a number indicating the hypothesised true value of the mean under the null. For the moment h0=0.
`lowParam`	numeric defining the smallest delta of the search space for the test-defining deltaS for scenario 3. Currently not yet in use.
`highParam`	numeric defining the largest delta of the search space for the test-defining deltaS for scenario 3. Currently not yet in use.
`tol`	a number that defines the stepsizes between the lowParam and highParam.
`inverseMethod`	logical, always `TRUE` for the moment.
`logging`	logical, if `TRUE`, then add invSToTThresh to output.
`paired`	logical, if `TRUE` then paired t-test.
`maxIter`	numeric > 0, the maximum number of iterations of adjustment to the candidate set from lowParam to highParam, if the minimum is not found.

Value

Returns an object of class 'safeDesign'. An object of class 'safeDesign' is a list containing at least the following components:

nPlan the planned sample size(s).

parameter the safe test defining parameter. Here deltaS.

esMin the minimal clinically relevant standardised effect size provided by the user.

alpha the tolerable type I error provided by the user.

beta the tolerable type II error provided by the user.

alternative any of "twoSided", "greater", "less" provided by the user.

testType any of "oneSample", "paired", "twoSample" provided by the user.

paired logical, `TRUE` if "paired", `FALSE` otherwise.

h0 the specified hypothesised value of the mean or mean difference depending on whether it was a one-sample or a two-sample test.

ratio default is 1. Different from 1, whenever testType equals "twoSample", then it defines ratio between the planned randomisation of condition 2 over condition 1.

lowN the smallest n of the search space for n provided by the user.

highN the largest n of the search space for n provided by the user.

lowParam the smallest delta of the search space for delta provided by the user.

highParam the largest delta of the search space for delta provided by the user.

tol the step size between lowParam and highParam provided by the user.

pilot `FALSE` (default) specified by the user to indicate that the design is not a pilot study.

call the expression with which this function is called.

Examples

```
designPilotSafeT(nPlan=30)
```

---

designPilotSafeZ *Designs a Safe Z-Test Based on Planned Samples nPlan*

Description

Designs a safe experiment for a prespecified tolerable type I error based on planned sample size(s), which are fixed ahead of time. Outputs a list that includes phiS, i.e., the safe test defining parameter.

Usage

```
designPilotSafeZ(
  nPlan,
  alternative = c("twoSided", "greater", "less"),
  alpha = 0.05,
  sigma = 1,
  h0 = 0,
  kappa = sigma,
  tol = 1e-05,
  paired = FALSE,
  parameter = NULL
)
```

Arguments

`nPlan`	optional numeric vector of length at most 2. When provided, it is used to find the safe test defining parameter phiS. Note that if the purpose is to plan based on nPlan alone, then both meanDiffMin and beta should be set to NULL.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`sigma`	numeric > 0 representing the assumed population standard deviation used for the test.
`h0`	numeric, represents the null hypothesis, default h0=0.
`kappa`	the true population standard deviation. Default kappa=sigma.
`tol`	a number that defines the stepsizes between the lowParam and highParam.
`paired`	logical, if `TRUE` then paired z-test.
`parameter`	optional test defining parameter. Default set to `NULL`.

Value

Returns a ’safeDesign’ object

nPlan the sample size(s) to plan for. Provided by the user.

parameter the safe test defining parameter. Here phiS.

esMin `NULL` no minimally clinically relevant effect size provided.

alpha the tolerable type I error provided by the user.

beta `NULL`, no tolerable type II error specified.

alternative any of "twoSided", "greater", "less" provided by the user.

testType any of "oneSample", "paired", "twoSample" effectively provided by the user.

paired logical, `TRUE` if "paired", `FALSE` otherwise.

sigma the assumed population standard deviation used for the test provided by the user.

kappa the true population standard deviation, typically, sigma=kappa.

ratio default is 1. Different from 1, whenever testType equals "twoSample", then it defines ratio between the planned randomisation of condition 2 over condition 1.

tol the step size between parameter values in the candidate space.

pilot logical, specifying whether it’s a pilot design.

call the expression with which this function is called.

Examples

```
designPilotSafeZ(nPlan=30, alpha = 0.05)
```

`designSafeLogrank` *Designs a Safe Logrank Test Experiment*

Description

A designed experiment requires (1) an anticipated number of events nEvents, or even better nPlan, the number of participants to be recruited in the study, and (2) the parameter of the safe test, i.e., thetaS. Provided with a clinically relevant minimal hazard ratio hrMin, this function outputs thetaS = hrMin as the safe test defining parameter in accordance to the GROW criterion. If a tolerable type II error beta is provided then nEvents can be sampled. The sampled nEvents is then the smallest nEvents for which hrMin is found with power of at least 1 - beta under optional stopping. If exact equal `FALSE`, then the computations exploit the local asymptotic normal approximation to sampling distribution of the logrank test derived by Schoenfeld (1981).

Usage

```
designSafeLogrank(
  hrMin = NULL,
  beta = NULL,
  nEvents = NULL,
  h0 = 1,
  alternative = c("twoSided", "greater", "less"),
  alpha = 0.05,
  ratio = 1,
  exact = TRUE,
  tol = 1e-05,
  m0 = 50000L,
  m1 = 50000L,
  nSim = 1000L,
  nBoot = 10000L,
  parameter = NULL,
  groupSizePerTimeFunction = returnOne,
  pb = TRUE,
  ...
)
```

Arguments

`hrMin`	numeric that defines the minimal relevant hazard ratio, the smallest hazard ratio that we want to detect.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both "n" and "phiS". Note that 1-beta defines the power.
`nEvents`	numeric > 0, targetted number of events.
`h0`	numeric > 0, represents the null hypothesis, default h0=1.

`alternative`	a character string specifying the alternative hypothesis, which must be one of "twoSided" (default),"greater" or "less". The alternative is pitted against the null hypothesis of equality of the survival distributions. More specifically, let lambda1 be the hazard rate of group 1 (i.e., placebo), and lambda2 the hazard ratio of group 2 (i.e., treatment), then the null hypothesis states that the hazard ratio theta = lambda2/lambda1 = 1. If alternative = "less", the null hypothesis is compared to theta < 1, thus, lambda2 < lambda1, that is, the hazard of group 2 (i.e., treatment) is less than that of group 1 (i.e., placebo), hence, the treatment is beneficial. If alternative = "greater", then the null hypothesis is compared to theta > 1, thus, lambda2 > lambda1, hence, harm.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`ratio`	numeric > 0 representing the randomisation ratio of condition 2 (Treatment) over condition 1 (Placebo), thus, m1/m0. Note that m1 and m0 are not used to specify ratio. Ratio is only used when `zApprox=TRUE`, which ignores m1 and m0.
`exact`	a logical indicating whether the design should be based on the exact safe logrank test based on the hypergeometric likelihood. Default is `TRUE`, if `FALSE` then the design is based on a safe z-test.
`tol`	a number that defines the stepsizes between the lowParam and highParam.
`m0`	Number of subjects in the control group 0/1 at the beginning of the trial, i.e., nPlan[1].
`m1`	Number of subjects in the treatment group 1/2 at the beginning of the trial, i.e., nPlan[2].
`nSim`	integer > 0, the number of simulations needed to compute power or the number of events for the exact safe logrank test under continuous monitoring
`nBoot`	integer > 0 representing the number of bootstrap samples to assess the accuracy of the approximation of power or nEvents for the exact safe logrank test under continuous monitoring
`parameter`	Numeric > 0, represents the safe tests defining thetaS. Default NULL so it's decided by the algorithm, typically, this equals hrMin, which corresponds to the GROW choice.
`groupSizePerTimeFunction`	A function without parameters and integer output. This function provides the number of events at each time step. For instance, if `rpois(1, 7)` leads to a random number of events at each time step.
`pb`	logical, if `TRUE`, then show progress bar.
`...`	further arguments to be passed to or from methods.

Value

Returns a safeDesign object that includes:

nEvents the anticipated number of events, either (1) specified by the user, or (2) computed based on beta and thetaMin.

parameter the parameter that defines the safe test. Here log(thetaS).

esMin the minimally clinically relevant hazard ratio specified by the user.

alpha the tolerable type I error provided by the user.

beta the tolerable type II error provided by the user.

alternative any of "twoSided", "greater", "less" provided by the user.

testType "logrank".

ratio default is 1. It defines the ratio between the planned randomisation of condition 2 over condition 1.

pilot `FALSE` to indicate that the design is not a pilot study.

call the expression with which this function is called.

References

Schoenfeld, D. (1981). The asymptotic properties of nonparametric tests for comparing survival distributions. Biometrika, 68(1), 316-319.

Examples

```
designSafeLogrank(hrMin=0.7)
designSafeLogrank(hrMin=0.7, zApprox=TRUE)
designSafeLogrank(hrMin=0.7, beta=0.3, nSim=10)
designSafeLogrank(hrMin=0.7, nEvents=190, nSim=10)
```

`designSafeT` *Designs a Safe Experiment to Test Means with a T Test*

Description

A designed experiment requires (1) a sample size nPlan to plan for, and (2) the parameter of the safe test, i.e., deltaS. If nPlan is provided, then only the safe test defining parameter deltaS needs to determined. That resulting deltaS leads to an (approximately) most powerful safe test. Typically, nPlan is unknown and the user has to specify (i) a tolerable type II error beta, and (ii) a clinically relevant minimal population standardised effect size deltaMin. The procedure finds the smallest nPlan for which deltaMin is found with power of at least 1 - beta.

Usage

```
designSafeT(
  deltaMin = NULL,
  beta = NULL,
  nPlan = NULL,
  alpha = 0.05,
  h0 = 0,
  alternative = c("twoSided", "greater", "less"),
  lowN = 3L,
  highN = 1000000L,
  lowParam = 0.01,
  highParam = 1.5,
  tol = 0.01,
  testType = c("oneSample", "paired", "twoSample"),
  ratio = 1,
  nSim = 1000L,
  nBoot = 1000L,
  parameter = NULL,
  pb = TRUE,
  seed = NULL,
  ...
)
```

Arguments

`deltaMin`	numeric that defines the minimal relevant standardised effect size, the smallest effect size that we would the experiment to be able to detect.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both the sample sizes and deltaS, which defines the test. Note that 1-beta defines the power.
`nPlan`	vector of max length 2 representing the planned sample sizes.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent of n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`h0`	a number indicating the hypothesised true value of the mean under the null. For the moment h0=0.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`lowN`	integer minimal sample size of the (first) sample when computing the power due to optional stopping. Default lowN is set 1.
`highN`	integer minimal sample size of the (first) sample when computing the power due to optional stopping. Default highN is set 1e6.
`lowParam`	numeric defining the smallest delta of the search space for the test-defining deltaS for scenario 3. Currently not yet in use.
`highParam`	numeric defining the largest delta of the search space for the test-defining deltaS for scenario 3. Currently not yet in use.
`tol`	a number that defines the stepsizes between the lowParam and highParam.
`testType`	either one of "oneSample", "paired", "twoSample".
`ratio`	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.
`nSim`	integer > 0, the number of simulations needed to compute power or the number of samples paths for the safe z test under continuous monitoring.

`nBoot`	integer > 0 representing the number of bootstrap samples to assess the accuracy of approximation of the power, the number of samples for the safe z test under continuous monitoring, or for the computation of the logarithm of the implied target.
`parameter`	optional test defining parameter. Default set to `NULL`.
`pb`	logical, if `TRUE`, then show progress bar.
`seed`	integer, seed number.
`...`	further arguments to be passed to or from methods, but mainly to perform do.calls.

Value

Returns an object of class 'safeDesign'. An object of class 'safeDesign' is a list containing at least the following components:

nPlan the planned sample size(s).

parameter the safe test defining parameter. Here deltaS.

esMin the minimal clinically relevant standardised effect size provided by the user.

alpha the tolerable type I error provided by the user.

beta the tolerable type II error provided by the user.

alternative any of "twoSided", "greater", "less" provided by the user.

testType any of "oneSample", "paired", "twoSample" provided by the user.

paired logical, `TRUE` if "paired", `FALSE` otherwise.

h0 the specified hypothesised value of the mean or mean difference depending on whether it was a one-sample or a two-sample test.

ratio default is 1. Different from 1, whenever testType equals "twoSample", then it defines ratio between the planned randomisation of condition 2 over condition 1.

lowN the smallest n of the search space for n provided by the user.

highN the largest n of the search space for n provided by the user.

lowParam the smallest delta of the search space for delta provided by the user.

highParam the largest delta of the search space for delta provided by the user.

tol the step size between lowParam and highParam provided by the user.

pilot `FALSE` (default) specified by the user to indicate that the design is not a pilot study.

call the expression with which this function is called.

Examples

```
designObj <- designSafeT(deltaMin=0.8, alpha=0.03, alternative="greater")
designObj

# "Scenario 1.a": Minimal clinically relevant standarised mean difference and tolerable type
# II error also known. Goal: find nPlan.
designObj <- designSafeT(deltaMin=0.8, alpha=0.03, beta=0.4, nSim=10, alternative="greater")
designObj
```

```
# "Scenario 2": Minimal clinically relevant standarised mean difference and nPlan known.
# Goal: find the power, hence, the type II error of the procedure under optional stopping.

designObj <- designSafeT(deltaMin=0.8, alpha=0.03, nPlan=16, nSim=10, alternative="greater")
designObj
```

---

designSafeTwoProportions

Designs a Safe Experiment to Test Two Proportions in Stream Data

---

Description

The design requires the number of observations one expects to collect in each group in each data block. I.e., when one expects balanced data, one could choose `na = nb = 1` and would be allowed to analyse the data stream each time a new observation in both groups has come in. The best results in terms of power are achieved when the data blocks are chosen as small as possible, as this allows for analysing and updating the safe test as often as possible, to fit the data best. Further, the design requires two out of the following three parameters to be known:

- the power one aims to achieve (`1 - beta`),
- the minimal relevant difference between the groups (`delta`)
- the number of blocks planned (`nBlocksPlan`),

where the unknown out of the three will be estimated. In the case of an exploratory "pilot" analysis, one can also only provide the number of blocks planned.

Usage

```
designSafeTwoProportions(
  na,
  nb,
  nBlocksPlan = NULL,
  beta = NULL,
  delta = NULL,
  alternativeRestriction = c("none", "difference", "logOddsRatio"),
  alpha = 0.05,
  pilot = "FALSE",
  hyperParameterValues = NULL,
  previousSafeTestResult = NULL,
  M = 1000,
  simThetaAMin = NULL,
  simThetaAMax = NULL
)
```

Arguments

`na`	number of observations in group a per data block
`nb`	number of observations in group b per data block
`nBlocksPlan`	planned number of data blocks collected
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both "nBlocksPlan" and "delta". Note that 1-beta defines the power.
`delta`	a priori minimal relevant divergence between group means b and a, either a numeric between -1 and 1 for no alternative restriction or a restriction on difference, or a real for a restriction on the log odds ratio.
`alternativeRestriction`	a character string specifying an optional restriction on the alternative hypothesis; must be one of "none" (default), "difference" (difference group mean b minus group b) or "logOddsRatio" (the log odds ratio between group means b and a).
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`pilot`	logical, specifying whether it's a pilot design.
`hyperParameterValues`	named list containing numeric values for hyperparameters betaA1, betaA2, betaB1 and betaB2, with betaA1 and betaB1 specifying the parameter equivalent to `shape1` in `stats::dbeta` for groups A and B, respectively, and betaA2 and betaB2 equivalent to `shape2`. By default chosen to optimize evidence collected over subsequent experiments (REGRET). Pass in the following format: `list(betaA1 = numeric1, betaA2 = numeric2, betaB1 = numeric3, betaB2 = numeric4)`.
`previousSafeTestResult`	optionally, a previous safe test result can be provided. The posterior of the hyperparameters of this test is then used for the hyperparameter settings. Default NULL.
`M`	number of simulations used to estimate power or nBlocksPlan. Default `1000`.
`simThetaAMin`	minimal event rate in control group to simulate nPlan or power for. Can be specified when specifically interested in planning studies for specific event rates. Default `NULL`, then the entire parameter space (possibly restricted by delta) is used for simulation.
`simThetaAMax`	maximal event rate in control group to simulate nPlan or power for. Default `NULL`.

Value

Returns a 'safeDesign' object that includes:

nPlan the sample size(s) to plan for. Computed based on beta and meanDiffMin, or provided by the user if known.

parameter the safe test defining parameter: here the hyperparameters.

esMin the minimally clinically relevant effect size provided by the user.

alpha the tolerable type I error provided by the user.

beta the tolerable type II error specified by the user.

alternative any of "twoSided", "greater", "less" based on the `alternativeRestriction` provided by the user.

testType here 2x2

pilot logical, specifying whether it's a pilot design.

call the expression with which this function is called.

Examples

```
#plan for an experiment to detect minimal difference of 0.6 with a balanced design
set.seed(3152021)
designSafeTwoProportions(na = 1,
                         nb = 1,
                         alpha = 0.1,
                         beta = 0.20,
                         delta = 0.6,
                         alternativeRestriction = "none",
                         M = 75)

#safe analysis of a pilot: number of samples already known
designSafeTwoProportions(na = 1,
                          nb = 1,
                          nBlocksPlan = 20,
                          pilot = TRUE)

#specify own hyperparameters
hyperParameterValues <- list(betaA1 = 10, betaA2 = 1, betaB1 = 1, betaB2 = 10)
designSafeTwoProportions(na = 1,
                         nb = 1,
                         alpha = 0.1,
                         beta = 0.20,
                         delta = 0.6,
                         hyperParameterValues = hyperParameterValues,
                         alternativeRestriction = "none",
                         M = 75)

#restrict range of proportions for estimating nPlan in the control group
designSafeTwoProportions(na = 1,
                         nb = 1,
                         beta = 0.20,
                         delta = 0.3,
                         alternativeRestriction = "none",
                         M = 75,
                         simThetaAMin = 0.1, simThetaAMax = 0.2)
```

designSafeZ *Designs a Safe Z Experiment*

Description

A designed experiment requires (1) a sample size nPlan to plan for, and (2) the parameter of the safe test, i.e., phiS. Provided with a clinically relevant minimal mean difference meanDiffMin, this function outputs phiS = meanDiffMin as the safe test defining parameter in accordance to the GROW criterion. If a tolerable type II error, i.e., beta, is provided then nPlan can be sampled. The sampled nPlan is then the smallest nPlan for which meanDiffMin can be found with power at least 1 - beta under optional stopping.

Usage

```
designSafeZ(
  meanDiffMin = NULL,
  beta = NULL,
  nPlan = NULL,
  alpha = 0.05,
  h0 = 0,
  alternative = c("twoSided", "greater", "less"),
  sigma = 1,
  kappa = sigma,
  tol = 1e-05,
  testType = c("oneSample", "paired", "twoSample"),
  ratio = 1,
  parameter = NULL,
  nSim = 1000L,
  nBoot = 1000L,
  pb = TRUE,
  grow = TRUE,
  ...
)
```

Arguments

`meanDiffMin`	numeric that defines the minimal relevant mean difference, the smallest population mean that we would like to detect.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both "n" and "phiS". Note that 1-beta defines the power.
`nPlan`	optional numeric vector of length at most 2. When provided, it is used to find the safe test defining parameter phiS. Note that if the purpose is to plan based on nPlan alone, then both meanDiffMin and beta should be set to NULL.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.

`h0`	numeric, represents the null hypothesis, default h0=0.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`sigma`	numeric > 0 representing the assumed population standard deviation used for the test.
`kappa`	the true population standard deviation. Default kappa=sigma.
`tol`	a number that defines the stepsizes between the lowParam and highParam.
`testType`	either one of "oneSample", "paired", "twoSample".
`ratio`	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.
`parameter`	optional test defining parameter. Default set to `NULL`.
`nSim`	integer > 0, the number of simulations needed to compute power or the number of samples paths for the safe z test under continuous monitoring.
`nBoot`	integer > 0 representing the number of bootstrap samples to assess the accuracy of approximation of the power, the number of samples for the safe z test under continuous monitoring, or for the computation of the logarithm of the implied target.
`pb`	logical, if `TRUE`, then show progress bar.
`grow`	logical, default set to `TRUE` so the grow safe test is used in the design.
`...`	further arguments to be passed to or from methods.

Value

Returns a safeDesign object that includes:

nPlan the sample size(s) to plan for. Computed based on beta and meanDiffMin, or provided by the user if known.

parameter the safe test defining parameter. Here phiS.

esMin the minimally clinically relevant effect size provided by the user.

alpha the tolerable type I error provided by the user.

beta the tolerable type II error specified by the user.

alternative any of "twoSided", "greater", "less" provided by the user.

testType any of "oneSample", "paired", "twoSample" effectively provided by the user.

paired logical, `TRUE` if "paired", `FALSE` otherwise.

sigma the assumed population standard deviation used for the test provided by the user.

kappa the true population standard deviation, typically, sigma=kappa.

ratio default is 1. Different from 1, whenever testType equals "twoSample", then it defines ratio between the planned randomisation of condition 2 over condition 1.

tol the step size between parameter values in the candidate space.

pilot logical, specifying whether it's a pilot design.

call the expression with which this function is called.

References

Grunwald, de Heide and Koolen (2019) "Safe Testing" <arXiv:1906.07801>

Examples

```
designObj <- designSafeZ(meanDiffMin=0.8, alpha=0.08, beta=0.01, alternative="greater")

#nPlan known:
designObj <- designSafeZ(nPlan = 100, alpha=0.05)
```

`extractNameFromArgs` *Helper function: Get all names as entered by the user*

Description

Helper function: Get all names as entered by the user

Usage

```
extractNameFromArgs(list, name)
```

Arguments

`list`	list from which the element needs retrieving
`name`	character string, name of the item that need retrieving

Value

returns a character string

`generateNormalData` *Generates Normally Distributed Data Depending on the Design*

Description

The designs supported are "oneSample", "paired", "twoSample".

Usage

```
generateNormalData(
  nPlan,
  nSim = 1000L,
  deltaTrue = NULL,
  muGlobal = 0,
  sigmaTrue = 1,
  paired = FALSE,
  seed = NULL,
  muTrue = NULL
)
```

Arguments

`nPlan`	vector of max length 2 representing the planned sample sizes.
`nSim`	the number of replications, that is, experiments with max samples nPlan.
`deltaTrue`	numeric, the value of the true standardised effect size (test-relevant parameter).
`muGlobal`	numeric, the true global mean of a paired or two-sample t-test. Its value should not matter for the test. This parameter is treated as a nuisance.
`sigmaTrue`	numeric > 0,the true standard deviation of the data. Its value should not matter for the test.This parameter treated is treated as a nuisance.
`paired`	logical, if `TRUE` then paired t-test.
`seed`	To set the seed for the simulated data.
`muTrue`	numeric representing the true mean for simulations with a z-test. Default `NULL`

Value

Returns a list of two data matrices contains at least the following components:

dataGroup1 a matrix of data dimension nSim by `nPlan[1]`.

dataGroup2 a matrix of data dimension nSim by `nPlan[2]`.

Examples

```
generateNormalData(20, 15, deltaTrue=0.3)
```

`generateSurvData` *Generate Survival Data which Can Be Analysed With the ‘survival‘ Package*

Description

Generate Survival Data which Can Be Analysed With the ‘survival‘ Package

Usage

```
generateSurvData(
  nP,
  nT,
  alpha = 1,
  lambdaP,
  lambdaT,
  seed = NULL,
  nDigits = 0,
  startTime = 1,
  endTime = 180,
  orderTime = TRUE,
  competeRatio = 0
)
```

Arguments

`nP`	integer > 0 representing the number of of patients in the placebo group.
`nT`	integer > 0 representing the number of of patients in the treatment group.
`alpha`	numeric > 0, representing the shape parameter of the Weibull distribution. If alpha=1, then data are generated from the exponential, i.e., constant hazard. For alpha > 1 the hazard increases, if alpha < 1, the hazard decreases.
`lambdaP`	The (relative) hazard of the placebo group.
`lambdaT`	The (relative) hazard of the treatment group.
`seed`	A seed number.
`nDigits`	numeric, the number of digits to round of the random time to
`startTime`	numeric, adds this to the random times. Default 1, so the startTime is not 0, which is the start time of `rweibull`.
`endTime`	The endtime of the experiment.
`orderTime`	logical, if `TRUE` then put the data set in increasing order
`competeRatio`	The ratio of the data that is due to competing risk.

Value

A data set with time, status and group.

Examples

```
generateSurvData(800, 800, alpha=1, lambdaP=0.008, lambdaT=0.008/2)
```

`getArgs` *Helper function: Get all arguments as entered by the user*

Description

Helper function: Get all arguments as entered by the user

Usage

```
getArgs()
```

Value

a list of variable names of class "call" that can be changed into names

`getNameAlternative` *Gets the Label of the Alternative Hypothesis*

Description

Helper function that outputs the alternative hypothesis of the analysis.

Usage

```
getNameAlternative(
  alternative = c("twoSided", "greater", "less"),
  testType,
  h0 = 0
)
```

Arguments

`alternative`	A character string. "twoSided", "greater", "less".
`testType`	A character string either "oneSample", "paired", "twoSample", "gLogrank", or "eLogrank".
`h0`	the value of the null hypothesis

Value

Returns a character string with the name of the analysis.

getNameTestType *Gets the Label of the Test*

Description

Helper function that outputs the name of the analysis.

Usage

```
getNameTestType(testType, parameterName)
```

Arguments

`testType`	A character string. For the t-tests: "oneSample", "paired", "twoSample".
`parameterName`	The name of the parameter to identify test performed

Value

Returns a character string with the name of the analysis.

isTryError *Checks Whether a Vector of Object Inherits from the Class 'try-error'*

Description

Checks whether any of the provided objects contains a try error.

Usage

```
isTryError(...)
```

Arguments

`...`	objects that need testing.

Value

Returns `TRUE` if there's some object that's a try-error, `FALSE` when all objects are not try-errors.

Examples

```
x <- 1
y <- "a"
z <- try(integrate(exp, -Inf, Inf))
isTryError(x, y)
isTryError(x, y, z)
```

logrankSingleEExact *Helper function computes single component of the exact logrank e-value*

Description

Helper function computes single component of the exact logrank e-value

Usage

```
logrankSingleEExact(obs0, obs1, y0, y1, thetaS, theta0 = 1, ...)
```

Arguments

obs0	integer, number of observations in the control group.
obs1	integer, number of observations in the treatment group.
y0	integer, total number of participants in the control group.
y1	integer, total number of participants in the treatment group.
thetaS	numeric > 0 represents the safe test defining (GROW) alternative hypothesis obtained from designSafeLogrank().
theta0	numeric > 0 represents the null hypothesis. Default theta0=1.
...	further arguments to be passed to or from methods.

Value

Returns a list containing at least the following components:

logP0 Log likelihood of Fisher's hypergeometric at the null

logEValueLess Log likelihood of Fisher's hypergeometric at the alternative

logEValueGreater Log likelihood of Fisher's hypergeometric at 1/alternative

Examples

```
#'
y0Vector <- c(5, 4, 3, 3, 2, 1)
y1Vector <- c(5, 5, 4, 2, 2, 0)
obs0Vector <- c(1, 1, 0, 1, 0, 1)
obs1Vector <- c(0, 0, 1, 0, 1, 0)

logEValueGreater <- logEValueLess <- vector("numeric", length(y0Vector))

for (i in seq_along(y0Vector)) {
  tempResult <- logrankSingleEExact(obs0=obs0Vector[i], obs1=obs1Vector[i],
                                    y0=y0Vector[i], y1=y1Vector[i],
                                    thetaS=0.7, theta0=1)
  logEValueLess[i] <- tempResult[["logEValueLess"]]
  logEValueGreater[i] <- tempResult[["logEValueGreater"]]
}

eValueLess <- exp(sum(logEValueLess))
eValueLess #1.116161
eValueGreater <- exp(sum(logEValueGreater))
eValueGreater # 0.7665818
eValue <- 1/2*eValueLess + 1/2*eValueGreater
eValue # 0.9413714
```

---

logrankSingleZ *Helper function computes single component of the logrank statistic*

Description

Helper function computes single component of the logrank statistic

Usage

```
logrankSingleZ(obs0, obs1, y0, y1, ...)
```

Arguments

obs0	integer, number of observations in the control group
obs1	integer, number of observations in the treatment group
y0	integer, total number of participants in the control group
y1	integer, total number of participants in the treatment group
...	further arguments to be passed to or from methods.

Value

Returns a list containing at least the following components:

oMinE observed minus expected.

v hypergeometric variance.

Examples

```
y0Vector <- c(6, 4, 4, 1, 0)
y1Vector <- c(6, 6, 5, 2, 2)
obs0Vector <- c(1, 0, 2, 1, 0)
obs1Vector <- c(0, 1, 1, 0, 1)

varVector <- oMinEVector <-y0Vector

for (i in seq_along(y0Vector)) {
  tempResult <- logrankSingleZ(obs0=obs0Vector[i], obs1=obs1Vector[i],
                              y0=y0Vector[i], y1=y1Vector[i])
  oMinEVector[i] <- tempResult[["oMinE"]]
  varVector[i] <- tempResult[["v"]]
}

sum(oMinEVector)/sqrt(sum(varVector))
```

`plot.safe2x2Sim` *Plots Results of Simulations for Comparing Hyperparameters for Safe Tests of Two Proportions*

Description

Plots Results of Simulations for Comparing Hyperparameters for Safe Tests of Two Proportions

Usage

```
## S3 method for class 'safe2x2Sim'
plot(x, ...)
```

Arguments

`x`	a result object obtained through `simulateTwoProportions()`.
`...`	further arguments to be passed to or from methods.

Value

Plot data, mainly called for side effects, the plot of simulation results.

Examples

```
priorList1 <- list(betaA1 = 10, betaA2 = 1, betaB1 = 1, betaB2 = 10)
priorList2 <- list(betaA1 = 0.18, betaA2 = 0.18, betaB1 = 0.18, betaB2 = 0.18)
priorList3 <- list(betaA1 = 1, betaA2 = 1, betaB1 = 1, betaB2 = 1)

simResult <- simulateTwoProportions(
  hyperparameterList = list(priorList1, priorList2, priorList3),
  alternativeRestriction = "none",
  alpha = 0.1, beta = 0.2, na = 1, nb = 1,
  deltamax = -0.4, deltamin = -0.9, deltaGridSize = 3,
  M = 10
  )

plot(simResult)
```

plot.safeTSim *Plots a 'safeTSim' Object*

Description

Plots a 'safeTSim' Object

Usage

```
## S3 method for class 'safeTSim'
plot(x, y = NULL, showOnlyNRejected = FALSE, nBin = 25, ...)
```

Arguments

`x`	a 'safeDesign' object acquired from `designSafeT()`.
`y`	`NULL`.
`showOnlyNRejected`	logical, when `TRUE` discards the cases that did not reject.
`nBin`	numeric > 0, the minimum number of bins in the histogram.
`...`	further arguments to be passed to or from methods.

Value

a histogram object, and called for its side-effect to plot the histogram.

Examples

```
# Design safe test
alpha <- 0.05
beta <- 0.20
designObj <- designSafeT(1, alpha=alpha, beta=beta)

# Design frequentist test
freqObj <- designFreqT(1, alpha=alpha, beta=beta)

# Simulate under the alternative with deltaTrue=deltaMin
simResults <- simulate(designObj, nSim=100)

plot(simResults)

plot(simResults, showOnlyNRejected=TRUE)
```

plotConfidenceSequenceTwoProportions

Plot bounds of a safe confidence sequence of the difference or log odds ratio for two proportions against the number of data blocks in two data streams ya and yb.

Description

Plot bounds of a safe confidence sequence of the difference or log odds ratio for two proportions against the number of data blocks in two data streams ya and yb.

Usage

```
plotConfidenceSequenceTwoProportions(
  ya,
  yb,
  safeDesign,
  differenceMeasure = c("difference", "odds"),
  precision = 100,
  deltaStart = 0.001,
  deltaStop = 3,
  trueDifference = NA
)
```

Arguments

`ya`	positive observations/ events per data block in group a: a numeric with integer values between (and including) 0 and `na`, the number of observations in group a per block.
`yb`	positive observations/ events per data block in group b: a numeric with integer values between (and including) 0 and `nb`, the number of observations in group b per block.
`safeDesign`	a safe test design for two proportions retrieved through `designSafeTwoProportions()`.
`differenceMeasure`	the difference measure to construct the confidence interval for: one of "difference" and "odds".
`precision`	precision of the grid to search over for the confidence sequence bounds.
`deltaStart`	for the odds difference measure: the (absolute value of the) smallest log odds ratio to assess for in- or exclusion in the confidence sequence. Default 0.001.
`deltaStop`	for the odds difference measure: the (absolute value of the) highest log odds ratio to assess for in- or exclusion in the confidence sequence. Default 3.
`trueDifference`	true difference or log odds ratio in groups A and B: added to the plot.

Value

no return value; called for its side effects, a plot of the confidence sequence.

Examples

```
set.seed(39413)
ya <- rbinom(n = 30, size = 1, prob = 0.1)
yb <- rbinom(n = 30, size = 1, prob = 0.8)
balancedSafeDesign <- designSafeTwoProportions(na = 1,
                                               nb = 1,
                                               nBlocksPlan = 30)
plotConfidenceSequenceTwoProportions(ya = ya,
                                     yb = yb,
                                     safeDesign = balancedSafeDesign,
                                     differenceMeasure = "difference",
                                     precision = 15,
                                     trueDifference = 0.7)

#log odds ratio difference measure
plotConfidenceSequenceTwoProportions(ya = ya,
                                     yb = yb,
                                     safeDesign = balancedSafeDesign,
                                     differenceMeasure = "odds",
                                     precision = 15,
                                     deltaStop = 5,
                                     trueDifference = log(36))

#switch ya and yb: observe negative log odds ratio in the data, plot mirrored in x-axis
plotConfidenceSequenceTwoProportions(ya = yb,
                                     yb = ya,
                                     safeDesign = balancedSafeDesign,
                                     differenceMeasure = "odds",
                                     precision = 15,
                                     deltaStop = 5,
                                     trueDifference = -log(36))
```

`plotHistogramDistributionStoppingTimes` *Plots the Histogram of Stopping Times*

Description

Helper function to display the histogram of stopping times.

Usage

```
plotHistogramDistributionStoppingTimes(
  safeSim,
  nPlan,
  deltaTrue,
  showOnlyNRejected = FALSE,
  nBin = 25L,
  ...
)
```

Arguments

`safeSim`	A safeSim object, returned from `replicateTTests`.
`nPlan`	numeric > 0, the planned sample size(s).
`deltaTrue`	numeric, that represents the true underlying standardised effect size delta.
`showOnlyNRejected`	logical, when `TRUE` discards the cases that did not reject.
`nBin`	numeric > 0, the minimum number of bins in the histogram.
`...`	further arguments to be passed to or from methods.

Value

a histogram object, and called for its side-effect to plot the histogram.

Examples

```
# Design safe test
alpha <- 0.05
beta <- 0.20
designObj <- designSafeT(1, alpha=alpha, beta=beta)

# Design frequentist test
freqObj <- designFreqT(1, alpha=alpha, beta=beta)

# Simulate under the alternative with deltaTrue=deltaMin
simResults <- replicateTTests(nPlan=designObj$nPlan, deltaTrue=1, parameter=designObj$parameter,
nPlanFreq=freqObj$nPlan)

plotHistogramDistributionStoppingTimes(
  simResults$safeSim, nPlan = simResults$nPlan,
  deltaTrue = simResults$deltaTrue)
```

`plotSafeTDesignSampleSizeProfile`
Plots the Sample Sizes Necessary for a Tolerable Alpha and Beta as a Function of deltaMin

---

Description

For given tolerable alpha and beta, (1) the planned sample sizes to using a safe test, (2) the frequentist test, and (3) the average sample size necessary due to optional stopping are plotted as a function of the minimal clinically relevant standardised effect size deltaMin.

Usage

```
plotSafeTDesignSampleSizeProfile(
  alpha = 0.05,
  beta = 0.2,
  nMax = 100,
  lowDeltaMin = 0.1,
  highDeltaMin = 1,
  stepDeltaMin = 0.1,
  testType = c("oneSample", "paired", "twoSample"),
  alternative = c("twoSided", "greater", "less"),
  ratio = 1,
  nSim = 1000L,
  nBoot = 1000L,
  seed = NULL,
  pb = TRUE,
  freqPlot = FALSE,
  ...
)
```

Arguments

`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent of n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both the sample sizes and deltaS, which defines the test. Note that 1-beta defines the power.
`nMax`	numeric, the maximum number of samples one has budget for to collect data.
`lowDeltaMin`	numeric, lowest value for deltaMin of interest
`highDeltaMin`	numeric, largest value for deltaMin of interest
`stepDeltaMin`	numeric, step size between lowDeltaMin and highDeltaMin
`testType`	either one of "oneSample", "paired", "twoSample".
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`ratio`	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.
`nSim`	integer > 0, the number of simulations needed to compute power or the number of samples paths for the safe z test under continuous monitoring.
`nBoot`	integer > 0 representing the number of bootstrap samples to assess the accuracy of approximation of the power, the number of samples for the safe z test under continuous monitoring, or for the computation of the logarithm of the implied target.
`seed`	integer, seed number.
`pb`	logical, if `TRUE`, then show progress bar.

`freqPlot`	logical, if `TRUE` plot frequentist sample size profiles.
`...`	further arguments to be passed to or from methods, but mainly to perform do.calls.

Value

Returns a list that contains the planned sample size needed for the frequentist and safe tests as a function of the minimal clinically relevant effect sizes. The returned list contains at least the following components:

alpha the tolerable type I error provided by the user.

beta the tolerable type II error provided by the user.

maxN the largest number of samples provided by the user.

deltaDomain vector of the domain of deltaMin.

allN1PlanFreq vector of the planned sample sizes needed for the frequentist test corresponding to alpha and beta.

allN1PlanSafe vector of the planned sample sizes needed for the safe test corresponding to alpha and beta.

allDeltaS vector of safe test defining deltaS.

Examples

```
plotSafeTDesignSampleSizeProfile(nSim=1e2L)
```

`print.safe2x2Sim` *Prints Results of Simulations for Comparing Hyperparameters for Safe Tests of Two Proportions*

Description

Prints Results of Simulations for Comparing Hyperparameters for Safe Tests of Two Proportions

Usage

```
## S3 method for class 'safe2x2Sim'
print(x, ...)
```

Arguments

`x`	a result object obtained through `simulateTwoProportions()`.
`...`	further arguments to be passed to or from methods.

Value

The data frame with simulation results, called for side effects to pretty print the simulation results.

Examples

```
priorList1 <- list(betaA1 = 10, betaA2 = 1, betaB1 = 1, betaB2 = 10)
priorList2 <- list(betaA1 = 0.18, betaA2 = 0.18, betaB1 = 0.18, betaB2 = 0.18)
priorList3 <- list(betaA1 = 1, betaA2 = 1, betaB1 = 1, betaB2 = 1)

simResult <- simulateTwoProportions(
  hyperparameterList = list(priorList1, priorList2, priorList3),
  alternativeRestriction = "none",
  alpha = 0.1, beta = 0.2, na = 1, nb = 1,
  deltamax = -0.4, deltamin = -0.9, deltaGridSize = 3,
  M = 10
  )
```

print.safeDesign *Print Method for Safe Tests*

Description

Printing objects of class 'safeTest' modelled after `print.power.htest()`.

Usage

```
## S3 method for class 'safeDesign'
print(x, digits = getOption("digits"), prefix = "\t", ...)
```

Arguments

`x`	a safeTest object.
`digits`	number of significant digits to be used.
`prefix`	string, passed to strwrap for displaying the method components.
`...`	further arguments to be passed to or from methods.

Value

No returned value, called for side effects.

Examples

```
designSafeZ(meanDiffMin=0.5)
designSafeT(deltaMin=0.5)
designSafeLogrank(hrMin=0.7)
```

print.safeTest *Print Method for Safe Tests*

Description

Printing objects of class 'safeTest' modelled after `print.htest()`.

Usage

```
## S3 method for class 'safeTest'
print(x, digits = getOption("digits"), prefix = "\t", ...)
```

Arguments

`x`	a safeTest object.
`digits`	number of significant digits to be used.
`prefix`	string, passed to strwrap for displaying the method components.
`...`	further arguments to be passed to or from methods.

Value

No returned value, called for side effects.

Examples

```
safeTTest(rnorm(19), pilot=TRUE)
safeZTest(rnorm(19), pilot=TRUE)
```

print.safeTSim *Prints a safeTSim Object*

Description

Prints a safeTSim Object

Usage

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

Arguments

`x`	a 'safeTSim' object.
`...`	further arguments to be passed to or from methods.

Value

No returned value, called for side effects.

Examples

```
designObj <- designSafeT(1, beta=0.2, nSim=10)

# Data under deltaTrue=deltaMin
simObj <- simulate(designObj, nSim=10)
print(simObj)
```

`replicateTTests` *Simulate Early Stopping Experiments*

Description

Simulate multiple data sets to show the effects of optional testing for safe (and frequentist) tests.

Usage

```
replicateTTests(
  nPlan,
  deltaTrue,
  muGlobal = 0,
  sigmaTrue = 1,
  paired = FALSE,
  alternative = c("twoSided", "greater", "less"),
  lowN = 3,
  nSim = 1000L,
  alpha = 0.05,
  beta = 0.2,
  safeOptioStop = TRUE,
  parameter = NULL,
  freqOptioStop = FALSE,
  nPlanFreq = NULL,
  logging = TRUE,
  seed = NULL,
  pb = TRUE,
  ...
)
```

Arguments

`nPlan`	vector of max length 2 representing the planned sample sizes.
`deltaTrue`	numeric, the value of the true standardised effect size (test-relevant parameter).
`muGlobal`	numeric, the true global mean of a paired or two-sample t-test. Its value should not matter for the test. This parameter is treated as a nuisance.

`sigmaTrue`	numeric > 0,the true standard deviation of the data. Its value should not matter for the test.This parameter treated is treated as a nuisance.
`paired`	logical, if `TRUE` then paired t-test.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`lowN`	integer that defines the smallest n of our search space for n.
`nSim`	the number of replications, that is, experiments with max samples nPlan.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent of n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control necessary to calculate both the sample sizes and deltaS, which defines the test. Note that 1-beta defines the power.
`safeOptioStop`	logical, `TRUE` implies that optional stopping simulation is performed for the safe test.
`parameter`	numeric, the safe test defining parameter, i.e., deltaS (use designSafeT to find this).
`freqOptioStop`	logical, `TRUE` implies that optional stopping simulation is performed for the frequentist test.
`nPlanFreq`	the frequentist sample size(s) to plan for. Acquired from `designFreqT()`.
`logging`	logical, if `TRUE`, then return the simulated data.
`seed`	To set the seed for the simulated data.
`pb`	logical, if `TRUE`, then show progress bar.
`...`	further arguments to be passed to or from methods.

Value

Returns an object of class "safeTSim". An object of class "safeTSim" is a list containing at least the following components:

nPlan the planned sample size(s).

deltaTrue the value of the true standardised effect size (test-relevant parameter) provided by the user.

muGlobal the true global mean of a paired or two-sample t-test (nuisance parameter) provided by the user.

paired if `TRUE` then paired t-test.

alternative any of "twoSided", "greater", "less" provided by the user.

lowN the smallest number of samples (first group) at which monitoring of the tests begins.

nSim the number of replications of the experiment.

alpha the tolerable type I error provided by the user.

beta the tolerable type II error provided by the user.

testType any of "oneSample", "paired", "twoSample" provided by the user.

parameter the parameter (point prior) used in the safe test derived from the design. Acquired from `designSafeT()`.

nPlanFreq the frequentist planned sample size(s). Acquired from `designFreqT()`

safeSim list with the simulation results of the safe test under optional stopping.

freqSim list with the simulation results of the frequentist test under optional stopping.

Examples

```
# Design safe test
alpha <- 0.05
beta <- 0.20
designObj <- designSafeT(1, alpha=alpha, beta=beta)

# Design frequentist test
freqObj <- designFreqT(1, alpha=alpha, beta=beta)

# Simulate under the alternative with deltaTrue=deltaMin
simResults <- replicateTTests(nPlan=designObj$nPlan, deltaTrue=1, parameter=designObj$parameter,
                              nPlanFreq=freqObj$nPlan, beta=beta, nSim=250)

# Should be about 1-beta
simResults$safeSim$powerAtN1Plan

# This is higher due to optional stopping
simResults$safeSim$powerOptioStop

# Optional stopping allows us to do better than n1PlanFreq once in a while
simResults$safeSim$probLeqN1PlanFreq
graphics::hist(simResults$safeSim$allN, main="Histogram of stopping times", xlab="n1",
               breaks=seq.int(designObj$nPlan[1]))

# Simulate under the alternative with deltaTrue > deltaMin
simResults <- replicateTTests(nPlan=designObj$nPlan, deltaTrue=1.5, parameter=designObj$parameter,
                              nPlanFreq=freqObj$nPlan, beta=beta, nSim=250)

# Should be larger than 1-beta
simResults$safeSim$powerAtN1Plan

# This is even higher due to optional stopping
simResults$safeSim$powerOptioStop

# Optional stopping allows us to do better than n1PlanFreq once in a while
simResults$safeSim$probLeqN1PlanFreq
graphics::hist(simResults$safeSim$allN, main="Histogram of stopping times", xlab="n1",
               breaks=seq.int(designObj$nPlan[1]))

# Under the null deltaTrue=0
simResults <- replicateTTests(nPlan=designObj$nPlan, deltaTrue=0, parameter=designObj$parameter,
                       nPlanFreq=freqObj$nPlan, freqOptioStop=TRUE, beta=beta, nSim=250)

# Should be lower than alpha, because if the null is true, P(S > 1/alpha) < alpha for all n
simResults$safeSim$powerAtN1Plan
```

```
# This is a bit higher due to optional stopping, but if the null is true,
# then still P(S > 1/alpha) < alpha for all n
simResults$safeSim$powerOptioStop

# Should be lowr than alpha, as the experiment is performed as was planned
simResults$freqSim$powerAtN1Plan

# This is larger than alpha, due to optional stopping.
simResults$freqSim$powerOptioStop
simResults$freqSim$powerOptioStop > alpha
```

`returnOne` *Auxiliary function for sampling of the logrank simulations to return the integer 1 event per time.*

Description

Auxiliary function for sampling of the logrank simulations to return the integer 1 event per time.

Usage

```
returnOne()
```

Value

1

Examples

```
returnOne()
```

`rLogrank` *Randomly samples from a logrank distribution*

Description

Draws a number of occurrences in group 1 (treatment) out of obsTotal number of occurrences.

Usage

```
rLogrank(n = 1, y0, y1, obsTotal, theta)
```

Arguments

`n`	integer, number of observations to be sampled.
`y0`	Size of the risk set of group 0 (Placebo).
`y1`	Size of the risk set of group 1 (Treatment).
`obsTotal`	Total number of observations.
`theta`	Odds of group 1 over group 0 (treatment over placebo).

Value

integer representing the number of occurrences in group 1 out of obsTotal number of occurrences.

Author(s)

Muriel Felipe Perez-Ortiz and Alexander Ly

Examples

```
rLogrank(y0=360, y1=89, obsTotal=12, theta=3.14)
```

`safeLogrankTest` *Safe Logrank Test*

Description

A safe test to test whether there is a difference between two survival curves. This function builds on the Mantel-Cox version of the logrank test.

Usage

```
safeLogrankTest(
  formula,
  designObj = NULL,
  ciValue = NULL,
  data = NULL,
  survTime = NULL,
  group = NULL,
  pilot = FALSE,
  exact = TRUE,
  computeZ = TRUE,
  ...
)

safeLogrankTestStat(
  z,
  nEvents,
  designObj,
  ciValue = NULL,
  dataNull = 1,
  sigma = 1
)
```

Arguments

`formula`	a formula expression as for other survival models, of the form Surv(time, status) ~ groupingVariable, see `Surv` for more details.
`designObj`	a safe logrank design obtained from `designSafeLogrank`.
`ciValue`	numeric, represents the ciValue-level of the confidence sequence. Default ciValue=NULL, and ciValue = 1 - alpha, where alpha is taken from the design object.
`data`	an optional data frame in which to interpret the variables occurring in survTime and group.
`survTime`	an optional survival time object of class 'Surv' created with `Surv`, or a name of a column in the data set of class 'Surv'. Does not need specifying if a formula is provided, therefore set to `NULL` by default.
`group`	an optional factor, a grouping variable. Currently, only two levels allowed. Does not need specifying if a formula is provided, therefore set to `NULL` by default.
`pilot`	a logical indicating whether a pilot study is run. If `TRUE`, it is assumed that the number of samples is exactly as planned. The default null h0=1 is used, alpha=0.05, and alternative="twoSided" is used. To change these default values, please use `designSafeLogrank`.
`exact`	a logical indicating whether the exact safe logrank test needs to be performed based on the hypergeometric likelihood. Default is `TRUE`, if `FALSE` then the safe z-test (for Gaussian data) applied to the logrank z-statistic is used instead.
`computeZ`	logical. If `TRUE` computes the logrank z-statistic. Default is `TRUE`.
`...`	further arguments to be passed to or from methods.
`z`	numeric representing the observed logrank z statistic.
`nEvents`	numeric > 0, observed number of events.
`dataNull`	numeric > 0, the null hypothesis corresponding to the z statistics. By default dataNull = 1 representing equality of the hazard ratio.
`sigma`	numeric > 0, scaling in the data.

Value

Returns an object of class 'safeTest'. An object of class 'safeTest' is a list containing at least the following components:

statistic the value of the summary, i.e., z-statistic or the e-value.

nEvents The number of observed events.

eValue the e-value of the safe test.

confSeq An anytime-valid confidence sequence.

estimate To be implemented: An estimate of the hazard ratio.

testType "logrank".

dataName a character string giving the name(s) of the data.

designObj an object of class "safeDesign" obtained from `designSafeLogrank`.

sumStats a list containing.the time of events, the progression of the risk sets and events.

call the expression with which this function is called.

Functions

- `safeLogrankTestStat()`: Safe Logrank Test based on Summary Statistic Z All provided data (i.e., z-scores) are assumed to be centred on a hazard ratio = 1, thus, log(hr) = 0 , and the proper (e.g., hypergeometric) scaling is applied to the data, so sigma = 1. The null hypothesis in the design object pertains to the population and is allowed to differ from log(theta) = 0.

Examples

```
# Example taken from survival::survdiff

designObj <- designSafeLogrank(hrMin=1/2)

ovData <- survival::ovarian
ovData$survTime <- survival::Surv(ovData$futime, ovData$fustat)

safeLogrankTest(formula=survTime~ rx, data=ovData, designObj=designObj)

safeLogrankTest(survTime=survTime, group=rx, data=ovData, designObj=designObj)

# Examples taken from coin::logrank_test
## Example data (Callaert, 2003, Tab. 1)
#'
callaert <- data.frame(
  time = c(1, 1, 5, 6, 6, 6, 6, 2, 2, 2, 3, 4, 4, 5, 5),
  group = factor(rep(0:1, c(7, 8)))
)

designObj <- designSafeLogrank(hrMin=1/2)

safeLogrankTest(survival::Surv(callaert$time)~callaert$group,
                designObj = designObj)

safeLogrankTest(survTime=survival::Surv(callaert$time),
                group=callaert$group, designObj = designObj)

result <- safeLogrankTest(survTime=survival::Surv(callaert$time),
                group=callaert$group, designObj = designObj)

result

##  Sequentially
# Greater
```

```
eValueGreater <- exp(cumsum(result$sumStats$logEValueGreater))
# Less
eValueLess <- exp(cumsum(result$sumStats$logEValueLess))

# twoSided
eValueTwoSided <- 1/2*eValueGreater+1/2*eValueLess

eValueTwoSided
result$eValue

###### Example switching between safe exact and safe Gaussian logrank test

designObj <- designSafeLogrank(0.8, alternative="less")

dat <- safestats::generateSurvData(300, 300, 2, 0.0065, 0.0065*0.8, seed=1)
survTime <- survival::Surv(dat$time, dat$status)

resultE <- safeLogrankTest(survTime ~ dat$group,
                           designObj = designObj)

resultG <- safeLogrankTest(survTime ~ dat$group,
                           designObj = designObj, exact=FALSE)

resultE
resultG

###### Example switching between safe exact and safe Gaussian logrank test other side

designObj <- designSafeLogrank(1/0.8, alternative="greater")

resultE <- safeLogrankTest(survTime ~ dat$group,
                           designObj = designObj)

resultG <- safeLogrankTest(survTime ~ dat$group,
                           designObj = designObj, exact=FALSE)

if (log(resultE$eValue) >= 0 && log(resultG$eValue) >= 0 )
  stop("one-sided wrong")
```

`safeTTest` *Safe Student's T-Test.*

Description

A safe t-test adapted from `t.test()` to perform one and two sample t-tests on vectors of data.

Usage

```
safeTTest(
  x,
  y = NULL,
  designObj = NULL,
  paired = FALSE,
  varEqual = TRUE,
  pilot = FALSE,
  alpha = NULL,
  alternative = NULL,
  ciValue = NULL,
  na.rm = FALSE,
  ...
)

safe.t.test(
  x,
  y = NULL,
  designObj = NULL,
  paired = FALSE,
  var.equal = TRUE,
  pilot = FALSE,
  alpha = NULL,
  alternative = NULL,
  ...
)
```

Arguments

`x`	a (non-empty) numeric vector of data values.
`y`	an optional (non-empty) numeric vector of data values.
`designObj`	an object obtained from `designSafeT()`, or `NULL`, when pilot equals `TRUE`.
`paired`	a logical indicating whether you want a paired t-test.
`varEqual`	a logical variable indicating whether to treat the two variances as being equal. For the moment, this is always `TRUE`.
`pilot`	a logical indicating whether a pilot study is run. If `TRUE`, it is assumed that the number of samples is exactly as planned.
`alpha`	numeric > 0 only used if pilot equals `TRUE`. If pilot equals `FALSE`, then the alpha of the design object is used instead in constructing the decision rule S > 1/alpha.
`alternative`	a character only used if pilot equals `TRUE`. If pilot equals `FALSE`, then the alternative specified by the design object is used instead.
`ciValue`	numeric is the ciValue-level of the confidence sequence. Default ciValue=NULL, and ciValue = 1 - alpha
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments to be passed to or from methods.
`var.equal`	a logical variable indicating whether to treat the two variances as being equal. For the moment, this is always `TRUE`.

Value

Returns an object of class "safeTest". An object of class "safeTest" is a list containing at least the following components:

statistic the value of the t-statistic.

n The realised sample size(s).

eValue the realised e-value from the safe test.

confSeq A safe confidence interval for the mean appropriate to the specific alternative hypothesis.

estimate the estimated mean or difference in means or mean difference depending on whether it a one- sample test or a two-sample test was conducted.

stderr the standard error of the mean (difference), used as denominator in the t-statistic formula.

testType any of "oneSample", "paired", "twoSample" provided by the user.

dataName a character string giving the name(s) of the data.

designObj an object of class "safeTDesign" obtained from `designSafeT()`.

call the expression with which this function is called.

Examples

```
designObj <- designSafeT(deltaMin=0.6, alpha=0.008, alternative="greater",
                         testType="twoSample", ratio=1.2)

set.seed(1)
x <- rnorm(100)
y <- rnorm(100)
safeTTest(x, y, designObj=designObj)      #0.2959334

safeTTest(1:10, y = c(7:20), pilot=TRUE)      # s = 658.69 > 1/alpha
designObj <- designSafeT(deltaMin=0.6, alpha=0.008, alternative="greater",
                         testType="twoSample", ratio=1.2)

set.seed(1)
x <- rnorm(100)
y <- rnorm(100)
safe.t.test(x, y, alternative="greater", designObj=designObj)      #0.2959334

safe.t.test(1:10, y = c(7:20), pilot=TRUE)      # s = 658.69 > 1/alpha
```

`safeTTestStat` *Computes E-Values Based on the T-Statistic*

Description

A summary stats version of `safeTTest()` with the data replaced by t, n1 and n2, and the design object by deltaS.

Usage

```
safeTTestStat(
  t,
  parameter,
  n1,
  n2 = NULL,
  alternative = c("twoSided", "less", "greater"),
  tDensity = FALSE,
  paired = FALSE,
  ...
)
```

Arguments

`t`	numeric that represents the observed t-statistic.
`parameter`	numeric this defines the safe test S, i.e., a likelihood ratio of t distributions with in the denominator the likelihood with delta = 0 and in the numerator an average likelihood defined by 1/2 time the likelihood at the non-centrality parameter sqrt(nEff)*parameter and 1/2 times the likelihood at the non-centrality parameter -sqrt(nEff)*parameter.
`n1`	integer that represents the size in a one-sample t-test, (n2=`NULL`). When n2 is not `NULL`, this specifies the size of the first sample for a two-sample test.
`n2`	an optional integer that specifies the size of the second sample. If it's left unspecified, thus, `NULL` it implies that the t-statistic is based on one-sample.
`alternative`	a character only used if pilot equals `TRUE`. If pilot equals `FALSE`, then the alternative specified by the design object is used instead.
`tDensity`	Uses the the representation of the safe t-test as the likelihood ratio of t densities.
`paired`	a logical indicating whether you want a paired t-test.
`...`	further arguments to be passed to or from methods.

Value

Returns a numeric that represent the e10, that is, the e-value in favour of the alternative over the null

Examples

```
safeTTestStat(t=1, n1=100, 0.4)
safeTTestStat(t=3, n1=100, parameter=0.3)
```

safeTTestStatAlpha *safeTTestStat() Subtracted with 1/alpha.*

Description

This is basically just safeTTestStat() - 1/alpha. This function is used for root finding for pilot designs.

Usage

```
safeTTestStatAlpha(
  t,
  parameter,
  n1,
  n2 = NULL,
  alpha,
  alternative = c("twoSided", "greater", "less"),
  tDensity = FALSE
)
```

Arguments

t	numeric that represents the observed t-statistic.
parameter	numeric this defines the safe test S, i.e., a likelihood ratio of t distributions with in the denominator the likelihood with delta = 0 and in the numerator an average likelihood defined by 1/2 time the likelihood at the non-centrality parameter sqrt(nEff)*parameter and 1/2 times the likelihood at the non-centrality parameter -sqrt(nEff)*parameter.
n1	integer that represents the size in a one-sample t-test, (n2=NULL). When n2 is not NULL, this specifies the size of the first sample for a two-sample test.
n2	an optional integer that specifies the size of the second sample. If it's left unspecified, thus, NULL it implies that the t-statistic is based on one-sample.
alpha	numeric > 0 only used if pilot equals TRUE. If pilot equals FALSE, then the alpha of the design object is used instead in constructing the decision rule S > 1/alpha.
alternative	a character only used if pilot equals TRUE. If pilot equals FALSE, then the alternative specified by the design object is used instead.
tDensity	Uses the the representation of the safe t-test as the likelihood ratio of t densities.

Value

Returns a numeric that represent the e10 - 1/alpha, that is, the e-value in favour of the alternative over the null - 1/alpha.

Examples

```
safeTTestStat(t=1, n1=100, 0.4)
safeTTestStat(t=3, n1=100, parameter=0.3)
```

safeTTestStatTDensity *safeTTestStat() based on t-densities*

Description

This is basically just safeTTestStat() - 1/alpha. This function is used for root finding for pilot designs.

Usage

```
safeTTestStatTDensity(
  t,
  parameter,
  nu,
  nEff,
  alternative = c("twoSided", "less", "greater"),
  paired = FALSE,
  ...
)
```

Arguments

`t`	numeric that represents the observed t-statistic.
`parameter`	numeric this defines the safe test S, i.e., a likelihood ratio of t distributions with in the denominator the likelihood with delta = 0 and in the numerator an average likelihood defined by 1/2 time the likelihood at the non-centrality parameter sqrt(nEff)*parameter and 1/2 times the likelihood at the non-centrality parameter -sqrt(nEff)*parameter.
`nu`	numeric > 0 representing the degrees of freedom.
`nEff`	numeric > 0 representing the effective sample size in a two-sample problem. For one-sample problems this is equal to the sample size.
`alternative`	a character only used if pilot equals `TRUE`. If pilot equals `FALSE`, then the alternative specified by the design object is used instead.
`paired`	a logical indicating whether you want a paired t-test.
`...`	further arguments to be passed to or from methods.

Value

Returns a numeric that represent the e10, that is, the e-value in favour of the alternative over the null.

Examples

```
safeTTestStat(t=1, n1=100, 0.4)
safeTTestStat(t=3, n1=100, parameter=0.3)
```

safeTwoProportionsTest *Perform a Safe Test for Two Proportions with Stream Data*

Description

Perform a safe test for two proportions (a 2x2 contingency table test) with a result object retrieved through the design function for planning an experiment to compare two proportions in this package, `designSafeTwoProportions()`.

Usage

```
safeTwoProportionsTest(
  ya,
  yb,
  designObj = NULL,
  wantConfidenceSequence = FALSE,
  ciValue = NULL,
  confidenceBoundGridPrecision = 20,
  logOddsConfidenceSearchBounds = c(0.01, 5),
  pilot = FALSE
)

safe.prop.test(
  ya,
  yb,
  designObj = NULL,
  wantConfidenceSequence = FALSE,
  ciValue = NULL,
  confidenceBoundGridPrecision = 20,
  logOddsConfidenceSearchBounds = c(0.01, 5),
  pilot = FALSE
)
```

Arguments

`ya`	positive observations/ events per data block in group a: a numeric with integer values between (and including) 0 and `na`, the number of observations in group a per block.
`yb`	positive observations/ events per data block in group b: a numeric with integer values between (and including) 0 and `nb`, the number of observations in group b per block.
`designObj`	a safe test design for two proportions retrieved through `designSafeTwoProportions()`.
`wantConfidenceSequence`	logical that can be set to true when the user wants a safe confidence sequence to be estimated.

`ciValue`	coverage of the safe confidence sequence; default `NULL`, if NULL calculated as `1 - designObj[["alpha"]]`.
`confidenceBoundGridPrecision`	integer specifying the grid precision used to search for the confidence bounds. Default 20.
`logOddsConfidenceSearchBounds`	vector of to positive doubles specifying the upper and lower bound of the grid to search over for finding the confidence bound for the logOddsRatio restriction. Default `(0.01, 5)`.
`pilot`	logical that can be set to true when performing an exploratory analysis without a `designObj`; only allows for `na = nb = 1`.

Value

Returns an object of class 'safeTest'. An object of class 'safeTest' is a list containing at least the following components:

n The realised sample size(s).

eValue the e-value of the safe test.

dataName a character string giving the name(s) of the data.

designObj an object of class "safeDesign" described in `designSafeTwoProportions()`.

Examples

```
#balanced design
yb <- c(1,0,1,1,1,0,1)
ya <- c(1,0,1,0,0,0,1)
safeDesign <- designSafeTwoProportions(na = 1,
                                       nb = 1,
                                       beta = 0.20,
                                       delta = 0.6,
                                       alternativeRestriction = "none",
                                       M = 1e1)
safeTwoProportionsTest(ya = ya, yb = yb, designObj = safeDesign)

#pilot
safeTwoProportionsTest(ya = ya, yb = yb, pilot = TRUE)

#unbalanced design
yb <- c(1,0,1,1,1,0,1)
ya <- c(2,2,1,2,0,2,2)
safeDesign <- designSafeTwoProportions(na = 2,
                                       nb = 1,
                                       beta = 0.20,
                                       delta = 0.6,
                                       alternativeRestriction = "none",
                                       M = 1e1)
safeTwoProportionsTest(ya = ya, yb = yb, designObj = safeDesign)
```

`safeZ10Inverse` *Computes the Inverse of the Two-Sided Safe Z-Test*

Description

This helper function is used in `designSafeZ()` to find parameter. The function is the (two-sided) inverse of 'safeZTestStat'.

Usage

```
safeZ10Inverse(parameter, nEff, sigma = 1, alpha = 0.05)
```

Arguments

`parameter`	optional test defining parameter. Default set to `NULL`.
`nEff`	numeric > 0, the effective sample size.
`sigma`	numeric, the assumed known standard deviation, default 1.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.

Value

A number that represents a z-value. The function's domain is the positive real line and the range is the real line, i.e., the outcome space of the z-statistic.

Examples

```
safeZ10Inverse(0.4, n=13)
```

`safeZTest` *Safe Z-Test*

Description

Safe one and two sample z-tests on vectors of data. The function is modelled after `t.test()`.

Usage

```
safeZTest(
  x,
  y = NULL,
  paired = FALSE,
  designObj = NULL,
  pilot = FALSE,
  ciValue = NULL,
  tol = 1e-05,
  na.rm = FALSE,
  ...
)

safe.z.test(
  x,
  y = NULL,
  paired = FALSE,
  designObj = NULL,
  pilot = FALSE,
  tol = 1e-05,
  ...
)
```

Arguments

`x`	a (non-empty) numeric vector of data values.
`y`	an optional (non-empty) numeric vector of data values.
`paired`	a logical indicating whether you want the paired z-test.
`designObj`	an object obtained from `designSafeZ()`, or `NULL`, when pilot is set to `TRUE`.
`pilot`	a logical indicating whether a pilot study is run. If `TRUE`, it is assumed that the number of samples is exactly as planned. The default null h0=1 is used, alpha=0.05, and alternative="twoSided" is used. To change these default values, please use `designSafeZ()`.
`ciValue`	numeric is the ciValue-level of the confidence sequence. Default ciValue=NULL, and ciValue = 1 - alpha
`tol`	numeric > 0, only used if pilot equals `TRUE`, as it then specifies the mesh used to find the test defining parameter to construct a pilot design object.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments to be passed to or from methods.

Value

Returns an object of class 'safeTest'. An object of class 'safeTest' is a list containing at least the following components:

statistic the value of the test statistic. Here the z-statistic.

n The realised sample size(s).

eValue the e-value of the safe test.

confInt To be implemented: a safe confidence interval for the mean appropriate to the specific alternative hypothesis.

estimate the estimated mean or difference in means or mean difference depending on whether it was a one- sample test or a two-sample test.

h0 the specified hypothesised value of the mean or mean difference depending on whether it was a one-sample or a two-sample test.

testType any of "oneSample", "paired", "twoSample" effectively provided by the user.

dataName a character string giving the name(s) of the data.

designObj an object of class "safeDesign" described in `designSafeZ()`.

call the expression with which this function is called.

Examples

```
designObj <- designSafeZ(meanDiffMin=0.6, alpha=0.008,
                         alternative="greater", testType="twoSample",
                         ratio=1.2)

set.seed(1)
x <- rnorm(100)
y <- rnorm(100)
safeZTest(x, y, designObj=designObj)      #

safeZTest(1:10, y = c(7:20), pilot=TRUE, alternative="less")     # s = 7.7543e+20 > 1/alpha
```

`safeZTestStat` *Computes E-Values Based on the Z-Statistic*

Description

Computes e-values using the z-statistic and the sample sizes only based on the test defining parameter phiS.

Usage

```
safeZTestStat(
  z,
  phiS,
  n1,
  n2 = NULL,
  alternative = c("twoSided", "less", "greater"),
  paired = FALSE,
  sigma = 1,
  ...
)
```

Arguments

`z`	numeric that represents the observed z-statistic.
`phiS`	numeric this defines the safe test S, i.e., a likelihood ratio of z distributions with in the denominator the likelihood with mean difference 0 and in the numerator an average likelihood defined by the likelihood at the parameter value. For the two sided case 1/2 at the parameter value and 1/2 at minus the parameter value.
`n1`	integer that represents the size in a one-sample z-test, (n2=`NULL`). When n2 is not `NULL`, this specifies the size of the first sample for a two-sample test.
`n2`	an optional integer that specifies the size of the second sample. If it's left unspecified, thus, `NULL` it implies that the z-statistic is based on one-sample.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`paired`	a logical, if `TRUE` ignores n2, and indicates that a paired z-test is performed.
`sigma`	numeric, the assumed known standard deviation, default 1.
`...`	further arguments to be passed to or from methods.

Value

Returns an e-value.

Examples

```
safeZTestStat(z=1, n1=100, phiS=0.4)
safeZTestStat(z=3, n1=100, phiS=0.3)
```

`sampleLogrankStoppingTimes`
Simulate stopping times for the exact safe logrank test

Description

Simulate stopping times for the exact safe logrank test

Usage

```
sampleLogrankStoppingTimes(
  hazardRatio,
  alpha = 0.05,
  alternative = c("twoSided", "less", "greater"),
  m0 = 50000L,
  m1 = 50000L,
  nSim = 1000L,
  groupSizePerTimeFunction = returnOne,
  parameter = NULL,
  nMax = Inf,
  pb = TRUE
)
```

Arguments

`hazardRatio`	numeric that defines the data generating hazard ratio with which data are sampled.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`alternative`	a character string specifying the alternative hypothesis, which must be one of "twoSided" (default),"greater" or "less". The alternative is pitted against the null hypothesis of equality of the survival distributions. More specifically, let lambda1 be the hazard rate of group 1 (i.e., placebo), and lambda2 the hazard ratio of group 2 (i.e., treatment), then the null hypothesis states that the hazard ratio theta = lambda2/lambda1 = 1. If alternative = "less", the null hypothesis is compared to theta < 1, thus, lambda2 < lambda1, that is, the hazard of group 2 (i.e., treatment) is less than that of group 1 (i.e., placebo), hence, the treatment is beneficial. If alternative = "greater", then the null hypothesis is compared to theta > 1, thus, lambda2 > lambda1, hence, harm.
`m0`	Number of subjects in the control group 0/1 at the beginning of the trial, i.e., nPlan[1].
`m1`	Number of subjects in the treatment group 1/2 at the beginning of the trial, i.e., nPlan[2].
`nSim`	integer > 0, the number of simulations needed to compute power or the number of events for the exact safe logrank test under continuous monitoring
`groupSizePerTimeFunction`	A function without parameters and integer output. This function provides the number of events at each time step. For instance, if `rpois(1, 7)` leads to a random number of events at each time step.
`parameter`	Numeric > 0, represents the safe tests defining thetaS. Default NULL so it's decided by the algorithm, typically, this equals hrMin, which corresponds to the GROW choice.
`nMax`	An integer. Once nEvents hits nMax the experiment terminates, if it didn't stop due to threshold crossing crossing already. Default set to Inf.
`pb`	logical, if `TRUE`, then show progress bar.

Value

a list with stoppingTimes and breakVector. Entries of breakVector are 0, 1. A 1 represents stopping due to exceeding nMax, and 0 due to 1/alpha threshold crossing, or running out of participants, which implies that the corresponding stopping time is Inf.

Author(s)

Muriel Felipe Perez-Ortiz and Alexander Ly

Examples

```
sampleLogrankStoppingTimes(0.7, nSim=10)
```

sampleStoppingTimesSafeT

Simulate stopping times for the safe z-test

Description

Simulate stopping times for the safe z-test

Usage

```
sampleStoppingTimesSafeT(
  deltaTrue,
  alpha = 0.05,
  alternative = c("twoSided", "less", "greater"),
  testType = c("oneSample", "paired", "twoSample"),
  nSim = 1000L,
  nMax = 1000,
  ratio = 1,
  lowN = 3L,
  parameter = NULL,
  seed = NULL,
  wantEValuesAtNMax = FALSE,
  pb = TRUE
)
```

Arguments

`deltaTrue`	numeric, the value of the true standardised effect size (test-relevant parameter). This argument is used by 'designSafeT()' with 'deltaTrue <- deltaMin'
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent of n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`testType`	either one of "oneSample", "paired", "twoSample".
`nSim`	integer > 0, the number of simulations needed to compute power or the number of samples paths for the safe z test under continuous monitoring.
`nMax`	integer > 0, maximum sample size of the (first) sample in each sample path.
`ratio`	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.
`lowN`	integer minimal sample size of the (first) sample when computing the power due to optional stopping. Default lowN is set 1.
`parameter`	optional test defining parameter. Default set to `NULL`.
`seed`	integer, seed number.

`wantEValuesAtNMax`	logical. If `TRUE` then compute eValues at nMax. Default `FALSE`.
`pb`	logical, if `TRUE`, then show progress bar.

Value

a list with stoppingTimes and breakVector. Entries of breakVector are 0, 1. A 1 represents stopping due to exceeding nMax, and 0 due to 1/alpha threshold crossing, which implies that in corresponding stopping time is Inf.

Examples

```
sampleStoppingTimesSafeT(0.7, nSim=10)
```

`sampleStoppingTimesSafeZ`

Simulate stopping times for the safe z-test

Description

Simulate stopping times for the safe z-test

Usage

```
sampleStoppingTimesSafeZ(
  meanDiffMin,
  alpha = 0.05,
  alternative = c("twoSided", "less", "greater"),
  sigma = 1,
  kappa = sigma,
  nSim = 1000L,
  nMax = 1000,
  ratio = 1,
  testType = c("oneSample", "paired", "twoSample"),
  parameter = NULL,
  wantEValuesAtNMax = FALSE,
  pb = TRUE
)
```

Arguments

`meanDiffMin`	numeric that defines the minimal relevant mean difference, the smallest population mean that we would like to detect.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.

`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`sigma`	numeric > 0 representing the assumed population standard deviation used for the test.
`kappa`	the true population standard deviation. Default kappa=sigma.
`nSim`	integer > 0, the number of simulations needed to compute power or the number of samples paths for the safe z test under continuous monitoring.
`nMax`	integer > 0, maximum sample size of the (first) sample in each sample path.
`ratio`	numeric > 0 representing the randomisation ratio of condition 2 over condition 1. If testType is not equal to "twoSample", or if nPlan is of length(1) then ratio=1.
`testType`	either one of "oneSample", "paired", "twoSample".
`parameter`	optional test defining parameter. Default set to `NULL`.
`wantEValuesAtNMax`	logical. If `TRUE` then compute eValues at nMax. Default `FALSE`.
`pb`	logical, if `TRUE`, then show progress bar.

Value

a list with stoppingTimes and breakVector. Entries of breakVector are 0, 1. A 1 represents stopping due to exceeding nMax, and 0 due to 1/alpha threshold crossing, which implies that in corresponding stopping time is Inf.

Examples

```
sampleStoppingTimesSafeZ(0.7, nSim=10)
```

`selectivelyContinueTTestCombineData`
Selectively Continue Experiments that Did Not Lead to a Null Rejection for a (Safe) T-Test

---

Description

Helper function used in the vignette.

Usage

```
selectivelyContinueTTestCombineData(
  oldValues,
  valuesType = c("eValues", "pValues"),
  designObj = NULL,
  alternative = c("twoSided", "greater", "less"),
  oldData,
  deltaTrue,
  alpha = NULL,
  n1Extra = NULL,
  n2Extra = NULL,
  seed = NULL,
  paired = FALSE,
  muGlobal = 0,
  sigmaTrue = 1,
  moreMainText = ""
)
```

Arguments

`oldValues`	vector of e-values or p-values.
`valuesType`	character string either "eValues" or "pValues".
`designObj`	a safeDesign object obtained from `designSafeT`, or `NULL` if valuesType equal "pValues".
`alternative`	a character string specifying the alternative hypothesis must be one of "twoSided" (default), "greater" or "less".
`oldData`	a list of matrices with names "dataGroup1" and "dataGroup2".
`deltaTrue`	numeric, the value of the true standardised effect size (test-relevant parameter).
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent of n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`n1Extra`	integer, that defines the additional number of samples of the first group. If `NULL` and valuesType equals "eValues", then n1Extra equals `designObj$nPlan[1]`.
`n2Extra`	optional integer, that defines the additional number of samples of the second group. If `NULL`, and valuesType equals "eValues", then n2Extra equals `designObj$nPlan[2]`.
`seed`	To set the seed for the simulated data.
`paired`	logical, if `TRUE` then paired t-test.
`muGlobal`	numeric, the true global mean of a paired or two-sample t-test. Its value should not matter for the test. This parameter is treated as a nuisance.
`sigmaTrue`	numeric > 0,the true standard deviation of the data. Its value should not matter for the test.This parameter treated is treated as a nuisance.
`moreMainText`	character, additional remarks in the title of the histogram.

Value

a list that includes the continued s or p-values based on the combined data, and a list of the combined data.

Examples

```
alpha <- 0.05
mIter <- 1000L

designObj <- designSafeT(deltaMin=1, alpha=alpha, beta=0.2, nSim=100)
```

```
oldData <- generateNormalData(nPlan=designObj$nPlan, deltaTrue=0, nSim=mIter, seed=1)

eValues <- vector("numeric", length=mIter)

for (i in seq_along(eValues)) {
  eValues[i] <- safeTTest(x=oldData$dataGroup1[i, ], designObj=designObj)$eValue
}

# First run: 8 false null rejections
sum(eValues > 1/alpha)

continuedSafe <- selectivelyContinueTTestCombineData(
  oldValues=eValues, designObj=designObj, oldData=oldData,
  deltaTrue=0, seed=2)

# Second run: 1 false null rejections
sum(continuedSafe$newValues > 1/alpha)

# Third run: 0 false null rejections
eValues <- continuedSafe$newValues
oldData <- continuedSafe$combinedData
continuedSafe <- selectivelyContinueTTestCombineData(
  oldValues=eValues, designObj=designObj, oldData=oldData,
  deltaTrue=0, seed=3)
sum(continuedSafe$newValues > 1/alpha)
```

`setSafeStatsPlotOptionsAndReturnOldOnes` *Sets 'safestats' Plot Options and Returns the Current Plot Options.*

Description

Sets 'safestats' Plot Options and Returns the Current Plot Options.

Usage

```
setSafeStatsPlotOptionsAndReturnOldOnes(...)
```

Arguments

`...`	further arguments to be passed to or from methods.

Value

Returns a list with the user specified plot options.

Examples

```
oldPar <- setSafeStatsPlotOptionsAndReturnOldOnes()
graphics::plot(1:10, 1:10)
setPar <- graphics::par(oldPar)
```

simulate.safeDesign *Simulate Early Stopping Experiments for the T Test*

Description

Applied to a 'safeDesign' object this function empirically shows the performance of safe experiments under optional stopping.

Usage

```
## S3 method for class 'safeDesign'
simulate(
  object,
  nsim = nSim,
  seed = NULL,
  deltaTrue = NULL,
  muGlobal = 0,
  sigmaTrue = 1,
  lowN = 3,
  safeOptioStop = TRUE,
  freqOptioStop = FALSE,
  nPlanFreq = NULL,
  logging = TRUE,
  pb = TRUE,
  nSim = 1,
  ...
)
```

Arguments

`object`	A safeDesign obtained obtained from `designSafeT()`.
`nsim`	integer, formally the number of iterations, but by default nsim=nSim
`seed`	integer, seed number.
`deltaTrue`	numeric, if NULL, then the minimally clinically relevant standardised effect size is used as the true data generating effect size deltaTrue.
`muGlobal`	numeric, the true global mean of a paired or two-sample t-test. Its value should not matter for the test. This parameter is treated as a nuisance.
`sigmaTrue`	numeric > 0,the true standard deviation of the data. Its value should not matter for the test.This parameter treated is treated as a nuisance.
`lowN`	integer that defines the smallest n of our search space for n.
`safeOptioStop`	logical, `TRUE` implies that optional stopping simulation is performed for the safe test.
`freqOptioStop`	logical, `TRUE` implies that optional stopping simulation is performed for the frequentist test.

`nPlanFreq`	the frequentist sample size(s) to plan for. Acquired from `designFreqT()`.
`logging`	logical, if `TRUE`, then return the simulated data.
`pb`	logical, if `TRUE`, then show progress bar.
`nSim`	integer, number of iterations.
`...`	further arguments to be passed to or from methods.

Value

Returns an object of class "safeTSim". An object of class "safeTSim" is a list containing at least the following components:

nPlan the planned sample size(s).

deltaTrue the value of the true standardised effect size (test-relevant parameter) provided by the user.

muGlobal the true global mean of a paired or two-sample t-test (nuisance parameter) provided by the user.

paired if `TRUE` then paired t-test.

alternative any of "twoSided", "greater", "less" provided by the user.

lowN the smallest number of samples (first group) at which monitoring of the tests begins.

nSim the number of replications of the experiment.

alpha the tolerable type I error provided by the user.

beta the tolerable type II error provided by the user.

testType any of "oneSample", "paired", "twoSample" provided by the user.

parameter the parameter (point prior) used in the safe test derived from the design. Acquired from `designSafeT()`.

nPlanFreq the frequentist planned sample size(s). Acquired from `designFreqT`()

safeSim list with the simulation results of the safe test under optional stopping.

freqSim list with the simulation results of the frequentist test under optional stopping.

Examples

```
# Design safe test
alpha <- 0.05
beta <- 0.20
deltaMin <- 1
designObj <- designSafeT(deltaMin, alpha=alpha, beta=beta, nSim=100)

# Design frequentist test
freqObj <- designFreqT(deltaMin, alpha=alpha, beta=beta)

# Simulate based on deltaTrue=deltaMin
simResultsDeltaTrueIsDeltaMin <- simulate(object=designObj, nSim=100)

# Simulate based on deltaTrue > deltaMin
simResultsDeltaTrueIsLargerThanDeltaMin <- simulate(
  object=designObj, nSim=100, deltaTrue=2)

# Simulate under the null deltaTrue = 0
simResultsDeltaTrueIsNull <- simulate(
  object=designObj, nSim=100, deltaTrue=0)

simulate(object=designObj, deltraTrue=0, nSim=100, freqOptioStop=TRUE,
         nPlanFreq=freqObj$nPlan)
```

---

simulateCoverageDifferenceTwoProportions

Simulate the coverage of a safe confidence sequence for differences between proportions for a given distribution and safe design.

Description

Simulate the coverage of a safe confidence sequence for differences between proportions for a given distribution and safe design.

Usage

```
simulateCoverageDifferenceTwoProportions(
  successProbabilityA,
  trueDelta,
  safeDesign,
  precision = 100,
  M = 1000,
  numberForSeed = NA
)
```

Arguments

`successProbabilityA`	probability of observing a success in group A.
`trueDelta`	difference in probability between group A and B.
`safeDesign`	a safe test design for two proportions retrieved through `designSafeTwoProportions()`.
`precision`	precision of the grid to search over for the confidence sequence bounds. Default 100.
`M`	number of simulations to carry out. Default 1000.
`numberForSeed`	number for seed to set, default NA.

Value

the proportion of simulations where the trueDelta was included in the confidence sequence.

Examples

```
balancedSafeDesign <- designSafeTwoProportions(na = 1,
                                               nb = 1,
                                               nBlocksPlan = 20)
simulateCoverageDifferenceTwoProportions(successProbabilityA = 0.2,
                                         trueDelta = 0,
                                         safeDesign = balancedSafeDesign,
                                         M = 100,
                                         precision = 20,
                                         numberForSeed = 1082021)
```

---

simulateIncorrectStoppingTimesFisher

Simulate incorrect optional stopping with fisher's exact test's p-value as the stopping rule.

Description

Simulate incorrect optional stopping with fisher's exact test's p-value as the stopping rule.

Usage

```
simulateIncorrectStoppingTimesFisher(
  thetaA,
  thetaB,
  alpha,
  na,
  nb,
  maxSimStoptime = 10000,
  M = 1000,
  numberForSeed = NULL
)
```

Arguments

`thetaA`	Bernoulli distribution parameter in group A
`thetaB`	Bernoulli distribution parameter in group B
`alpha`	Significance level
`na`	number of observations in group a per data block
`nb`	number of observations in group b per data block
`maxSimStoptime`	maximal number of blocks to sample in each experiment
`M`	Number of simulations to carry out, default 1e3.
`numberForSeed`	number for seed to set, default NULL.

Value

list with stopping times and rejection decisions.

Examples

```
simulateIncorrectStoppingTimesFisher(thetaA = 0.3,
                                     thetaB = 0.3,
                                     alpha = 0.05,
                                     na = 1,
                                     nb = 1,
                                     M = 10,
                                     maxSimStoptime = 100,
                                     numberForSeed = 251)
```

`simulateOptionalStoppingScenarioTwoProportions`
Simulate an optional stopping scenario according to a safe design for two proportions

---

Description

Simulate an optional stopping scenario according to a safe design for two proportions

Usage

```
simulateOptionalStoppingScenarioTwoProportions(safeDesign, M, thetaA, thetaB)
```

Arguments

`safeDesign`	a 'safeDesign' object obtained through `designSafeTwoProportions()`.
`M`	integer, the number of data streams to sample.
`thetaA`	Bernoulli distribution parameter in group A
`thetaB`	Bernoulli distribution parameter in group B

Value

list with the simulation results of the safe test under optional stopping with the following components:

powerOptioStop Proportion of sequences where H0 was rejected

nMean Mean stopping time

probLessNDesign Proportion of experiments stopped before nBlocksPlan was reached

lowN Minimum stopping time

eValues All achieved E values

allN All stopping times

allSafeDecisions Decisions on rejecting H0 for each M

allRejectedN Stopping times of experiments where H0 was rejected

Examples

```
balancedSafeDesign <- designSafeTwoProportions(na = 1,
                                               nb = 1,
                                               nBlocksPlan = 30)
optionalStoppingSimulationResult <- simulateOptionalStoppingScenarioTwoProportions(
  safeDesign = balancedSafeDesign,
  M = 1e2,
  thetaA = 0.2,
  thetaB = 0.5
)
```

---

simulateTwoProportions *Compare Different Hyperparameter Settings for Safe Tests of Two Proportions.*

Description

Simulates for a range of divergence parameter values (differences or log odds ratios) the worst-case stopping times (i.e., number of data blocks collected) and expected stopping times needed to achieve the desired power for each hyperparameter setting provided.

Usage

```
simulateTwoProportions(
  hyperparameterList,
  alternativeRestriction = c("none", "difference", "logOddsRatio"),
  deltaDesign = NULL,
  alpha,
  beta,
  na,
  nb,
  deltamax = 0.9,
  deltamin = 0.1,
  deltaGridSize = 8,
  M = 100,
  maxSimStoptime = 10000,
  thetaAgridSize = 8
)
```

Arguments

`hyperparameterList`
: list object, its components hyperparameter lists with a format as described in `designSafeTwoProportions()`.

`alternativeRestriction`	a character string specifying an optional restriction on the alternative hypothesis; must be one of "none" (default), "difference" (difference group mean b minus group b) or "logOddsRatio" (the log odds ratio between group means b and a).
`deltaDesign`	optional; when using a restricted alternative, the value of the divergence measure used. Either a numeric between -1 and 1 for a restriction on difference, or a real for a restriction on the log odds ratio.
`alpha`	numeric in (0, 1) that specifies the tolerable type I error control –independent on n– that the designed test has to adhere to. Note that it also defines the rejection rule e10 > 1/alpha.
`beta`	numeric in (0, 1) that specifies the tolerable type II error control in the study. Necessary to calculate the worst case stopping time.
`na`	number of observations in group a per data block
`nb`	number of observations in group b per data block
`deltamax`	maximal effect size to calculate power for; between -1 and 1 for designs without restriction or a restriction on difference; real number for a restriction on the log odds ratio. Default `0.9`.
`deltamin`	minimal effect size to calculate power for; between -1 and 1 for designs without restriction or a restriction on difference; real number for a restriction on the log odds ratio. Default `0.1`.
`deltaGridSize`	numeric, positive integer: size of grid of delta values worst case and expected sample sizes are simulated for.
`M`	number of simulations used to estimate sample sizes. Default `100`.
`maxSimStoptime`	maximal stream length in simulations; when the e value does not reach the rejection threshold before the end of the stream, the maximal stream length is returned as the stopping time. Default `1e4`.
`thetaAgridSize`	numeric, positive integer: size of the grid of probability distributions examined for each delta value to find the worst case sample size over.

Value

Returns an object of class "safe2x2Sim". An object of class "safe2x2Sim" is a list containing at least the following components:

simDataA data frame containing simulation results with worst case and expected stopping times for each hyperparameter setting, for the specified or default range of effect sizes.

alpha the significance threshold used in the simulations

beta the type-II error control used in the simulations

deltaDesign the value of restriction on the alternative hypothesis parameter space used for the E variables in the simulations

restriction the type of restriction used for the E variables in the simulation

hyperparameters list of the hyperparameters tested in the simulation

Examples

```
priorList1 <- list(betaA1 = 10, betaA2 = 1, betaB1 = 1, betaB2 = 10)
priorList2 <- list(betaA1 = 0.18, betaA2 = 0.18, betaB1 = 0.18, betaB2 = 0.18)
priorList3 <- list(betaA1 = 1, betaA2 = 1, betaB1 = 1, betaB2 = 1)

simResult <- simulateTwoProportions(
  hyperparameterList = list(priorList1, priorList2, priorList3),
  alternativeRestriction = "none",
  alpha = 0.1, beta = 0.2, na = 1, nb = 1,
  deltamax = -0.4, deltamin = -0.9, deltaGridSize = 3,
  M = 10
  )

print(simResult)
plot(simResult)
```

`tryOrFailWithNA` *Tries to Evaluate an Expression and Fails with* `NA`

Description

The evaluation fails with `NA` by default, but it is also able to fail with other values.

Usage

```
tryOrFailWithNA(expr, value = NA_real_)
```

Arguments

`expr`	Expression to be evaluated.
`value`	Return value if there is an error, default is `NA_real_`.

Value

Returns the evaluation of the expression, or `value` if it doesn't work out.

Index