Package: optconerrf (via r-universe)

November 11, 2026

Title Optimal Monotone Conditional Error Functions

Version 1.0.3

Description Design and analysis of confirmatory adaptive clinical trials using the optimal conditional error framework according to Brannath and Bauer (2004) <doi:10.1111/j.0006-341X.2004.00221.x>. An extension to the optimal conditional error function using interim estimates as described in Brannath and Dreher (2024) <doi:10.48550/arXiv.2402.00814> and functions to ensure that the resulting conditional error function is non-increasing are also available.

Imports ggplot2, methods

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/morten-dreher/optconerrf

BugReports https://github.com/morten-dreher/optconerrf/issues

Config/roxygen2/version 8.1.0

Repository https://morten-dreher.r-universe.dev

Date/Publication 2026-09-03 20:16:15 UTC

RemoteUrl https://github.com/morten-dreher/optconerrf

RemoteRef HEAD

RemoteSha 4052f060a29b3c90d5e95c8715e5a66dc4ff304d

Contents

.rangeCheck . . . 2
getConstraintC_max . . . 3
getConstraintC_min . . . 3
getDesignOptimalConditionalErrorFunction . . . 4
getExpectedSecondStageInformation . . . 10
getLevelConstant . . . 12
getLikelihoodRatio . . . 13
getMonotoneFunction . . . 15
getMonotonisationConstants . . . 16
getNu . . . 17
getNuPrime . . . 18
getOptimalConditionalError . . . 19
getOverallPower . . . 20
getPsi . . . 21
getQ . . . 22
getSecondStageInformation . . . 23
getSimulationResults . . . 24
plot.TrialDesignOptimalConditionalError . . . 25
PowerResultsOptimalConditionalError . . . 26
print.PowerResultsOptimalConditionalError . . . 26
print.SimulationResultsOptimalConditionalError . . . 27
print.TrialDesignOptimalConditionalError . . . 27
SimulationResultsOptimalConditionalError . . . 28
summary.TrialDesignOptimalConditionalError . . . 28
TrialDesignOptimalConditionalError . . . 29

Index **30**

`.rangeCheck` *Simple range check for numeric variables*

Description

This function performs very basic range checks for numeric variables and throws an error if the range is violated. A custom hint may be added to the message.

Usage

```
.rangeCheck(variable, range, allowedEqual, hint = "")
```

Arguments

`variable`	The (named) variable to be checked.
`range`	A vector of length 2 giving the minimum and maximum of the allowed range.
`allowedEqual`	Logical. Are the borders of range valid values?
`hint`	Additional message that may be printed after the error.

Value

Invisibly returns TRUE if the check was successful.

---

getConstraintC_max *Calculates the upper constraint.*

Description

Calculates the upper constraint.

Usage

```
getConstraintC_max(firstStagePValue, design)
```

Arguments

firstStagePValue	First-stage p-value or p-values. Must be a numeric vector between 0 and 1.
design	An object of class TrialDesignOptimalConditionalError created by getDesignOptimalConditional Contains all necessary arguments to calculate the optimal conditional error function for the specified case.

Details

Internal function that calculates the upper constraint.

Value

Value of the upper constraint

---

getConstraintC_min *Calculates the lower constraint.*

Description

Calculates the lower constraint.

Usage

```
getConstraintC_min(firstStagePValue, design)
```

Arguments

`firstStagePValue`
: First-stage p-value or p-values. Must be a numeric vector between 0 and 1.

`design`
: An object of class `TrialDesignOptimalConditionalError` created by `getDesignOptimalConditional`. Contains all necessary arguments to calculate the optimal conditional error function for the specified case.

Details

Internal function that calculates the lower constraint.

Value

Value of the lower constraint

`getDesignOptimalConditionalErrorFunction` *Create a design object for the optimal conditional error function.*

Description

This function returns a design object which contains all important parameters for the specification of the optimal conditional error function. The returned object is of class `TrialDesignOptimalConditionalError` and can be passed to other package functions.

Usage

```
getDesignOptimalConditionalErrorFunction(
  alpha,
  alpha1,
  alpha0,
  conditionalPower = NA_real_,
  delta1 = NA_real_,
  delta1Min = NA_real_,
  delta1Max = Inf,
  ncp1 = NA_real_,
  ncp1Min = NA_real_,
  ncp1Max = Inf,
  useInterimEstimate = TRUE,
  firstStageInformation,
  likelihoodRatioDistribution,
  minimumSecondStageInformation = 0,
  maximumSecondStageInformation = Inf,
  minimumConditionalError = 0,
  maximumConditionalError = 1,
  conditionalPowerFunction = NA,
```

```
  levelConstantMinimum = 0,
  levelConstantMaximum = 10,
  enforceMonotonicity = TRUE,
  ...
)
```

Arguments

alpha	The overall type I error rate α of the design. Must be a numeric value between 0 and 1.
alpha1	Stage 1 efficacy boundary α_1 (p-value scale). Must be a numeric value between 0 and 1. Should be smaller than alpha0.
alpha0	Binding stage 1 futility boundary α_0 (p-value scale). Must be a numeric value between 0 and 1. Should be greater than alpha1. For use of a non-binding futility boundary, specify alpha0=1.
conditionalPower	The target conditional power CP of the design. Must be a numeric value.
delta1	Fixed effect assumption at which the conditional power should be achieved, expressed on the mean difference scale. Is only used if useInterimEstimate=FALSE. Alternatively, ncp1 can be specified. Must be a numeric value greater than 0.
delta1Min	The minimum for an interim estimate of the treatment effect, specified on the mean difference scale. If the interim estimate (on the mean difference scale) yields a value smaller than delta1Min, delta1Min is used for it. Is only used if useInterimEstimate=TRUE. Alternatively, ncp1Min can be specified. Must be a numeric value.
delta1Max	The maximum for an interim estimate of the treatment effect, specified on the mean difference scale. If the interim estimate (on the mean difference scale) yields a value larger than delta1Max, delta1Max is used for it. Is only used if useInterimEstimate=TRUE. Alternatively, ncp1Max can be specified. Must be a numeric value. Default value is Inf, i.e., no upper restriction.
ncp1	Fixed effect assumption at which the conditional power should be achieved, expressed on the non-centrality parameter scale. Is only used if useInterimEstimate=FALSE. Alternatively, delta1 can be specified. Must be a numeric value greater than 0.
ncp1Min	The minimum for an interim estimate of the treatment effect, specified on the non-centrality parameter scale. If the interim estimate (on the non-centrality parameter scale) yields a value smaller than ncp1Min, ncp1Min is used for it. Is only used if useInterimEstimate=TRUE. Alternatively, delta1Min can be specified. Must be a numeric value.
ncp1Max	The maximum for an interim estimate of the treatment effect, specified on the non-centrality parameter scale. If the interim estimate (on the non-centrality parameter scale) yields a value larger than ncp1Max, ncp1Max is used for it. Is only used if useInterimEstimate=TRUE. Alternatively, delta1Max can be specified. Must be a numeric value. Default value is Inf, i.e., no upper restriction.
useInterimEstimate	Logical. Defines whether or not an interim estimate should be used for conditional power. If TRUE, a lower cut-off for the interim estimate must be specified by delta1Min or ncp1Min. An upper cut-off may also be specified by

delta1Max or ncp1Max. If FALSE, the fixed effect size must be specified by delta1 or ncp1.

firstStageInformation

Information of the first stage of the trial. Must be a positive numeric value.

likelihoodRatioDistribution

The distribution to be used for the effect size of the likelihood ratio in the optimal conditional error function. Options are "fixed", "normal", "exp", "unif", "maxlr" for fixed effect size, normally distributed, exponentially distributed, uniformly distributed prior of the effect size and maximum likelihood ratio, respectively. Each case requires different additional specifications:

- likelihoodRatioDistribution="fixed" uses one (or more) fixed effect sizes for the likelihood ratio and requires the parameter deltaLR which provides the mean difference under which to calculate the likelihood ratio. If deltaLR contains multiple values, they may be weighted using an additional argument weightsDeltaLR. Omitting weightsDeltaLR automatically leads to equal weighting.
- likelihoodRatioDistribution="normal" uses a normal prior for the effect size and requires parameters deltaLR and tauLR for the mean and standard deviation of the normal distribution (both on mean difference scale).
- likelihoodRatioDistribution="exp" uses an exponential prior for the effect size and requires the parameter kappaLR which is the mean of the exponential distribution (on the mean difference scale).
- likelihoodRatioDistribution="unif" uses a uniform prior for the effect size and requires the specification of deltaMaxLR, which is the maximum of the support for the uniform likelihood ratio distribution (on the mean difference scale).
- likelihoodRatioDistribution="maxlr" estimates the non-centrality parameter to be used for the likelihood ratio from the data. No additional parameters must be specified.

minimumSecondStageInformation

The minimum information allowed in the second stage of the trial. Must be a numeric value. Default value is 0, i.e., no restriction.

maximumSecondStageInformation

The maximum information allowed in the second stage of the trial. Must be a numeric value. Default value is Inf, i.e., no restriction.

minimumConditionalError

Lower boundary for the optimal conditional error function. Default 0 (no restriction).

maximumConditionalError

Upper boundary for the optimal conditional error function. Default value is 1, however, the optimal conditional error function is inherently bounded by the conditional power.

conditionalPowerFunction

A user-specified function which calculates the conditional power from the first-stage p-value. This function should not be increasing in the first-stage p-value or monotonicity issues may occur.

`levelConstantMinimum`
: The minimum of the interval on which the value for the level constant should be searched. Default value is 0.

`levelConstantMaximum`
: The maximum of the interval on which the value for the level constant should be searched. Default value is 10.

`enforceMonotonicity`
: Logical. Determines whether or not the optimal conditional error function should automatically be modified to be non-increasing. Default is `TRUE`.

`...`
: Additional arguments required for the specification of the likelihood ratio.

Details

The design object contains the information required to determine the specific setting of the optimal conditional error function and can be passed to other package functions. From the given user specifications, the constant to achieve level condition for control of the overall type I error rate as well as the constants to ensure a non-increasing optimal CEF (if required) are automatically calculated.

Value

An object of class `TrialDesignOptimalConditionalError`, which can be passed to other package functions.

Likelihood ratio distribution

To calculate the optimal conditional error function, an assumption about the true parameter under which the second-stage information is to be minimised is required. Various options are available and can be specified via the argument `likelihoodRatioDistribution`:

- `likelihoodRatioDistribution="fixed"`: calculates the likelihood ratio for a fixed Δ. The non-centrality parameter of the likelihood ratio ϑ is then computed as `deltaLR*sqrt(firstStageInformation)` and the likelihood ratio is calculated as:

$$l(p_1) = e^{\Phi^{-1}(1-p_1)\vartheta - \vartheta^2/2}.$$

  `deltaLR` may also contain multiple elements, in which case a weighted likelihood ratio is calculated for the given values. Unless positive weights that sum to 1 are provided by the argument `weightsDeltaLR`, equal weights are assumed.

- `likelihoodRatioDistribution="normal"`: calculates the likelihood ratio for a normally distributed prior of ϑ with mean `deltaLR*sqrt(firstStageInformation)` (μ) and standard deviation `tauLR*sqrt(firstStageInformation)` (σ). The parameters `deltaLR` and `tauLR` must be specified on the mean difference scale.

$$l(p_1) = (1+\sigma^2)^{-\frac{1}{2}} \cdot e^{-(\mu/\sigma)^2/2 + (\sigma\Phi^{-1}(1-p_1)+\mu/\sigma)^2/(2\cdot(1+\sigma^2))}$$

- `likelihoodRatioDistribution="exp"`: calculates the likelihood ratio for an exponentially distributed prior of ϑ with mean `kappaLR*sqrt(firstStageInformation)` (η). The likelihood ratio is then calculated as:

$$l(p_1) = \kappa \cdot \sqrt{2\pi} \cdot e^{(\Phi^{-1}(1-p_1)-\eta)^2/2} \cdot \Phi(\Phi^{-1}(1-p_1) - \eta)$$

- `likelihoodRatioDistribution="unif"`: calculates the likelihood ratio for a uniformly distributed prior of ϑ on the support $[0, \Delta \cdot \sqrt{I_1}]$, where Δ is specified as `deltaMaxLR` and I_1 is the `firstStageInformation`.

$$l(p_1) = \frac{\sqrt{2\pi}}{\Delta \cdot \sqrt{I_1}} \cdot e^{\Phi^{-1}(1-p_1)^2/2} \cdot (\Phi(\Delta \cdot \sqrt{I_1} - \Phi^{-1}(1-p_1)) - p_1)$$

- `likelihoodRatioDistribution="maxlr"`: the non-centrality parameter ϑ is estimated from the data and no additional parameters must be specified. The likelihood ratio is estimated from the data as:

$$l(p_1) = e^{max(0,\Phi^{-1}(1-p_1))^2/2}$$

  The maximum likelihood ratio is always restricted to effect sizes $\vartheta \geq 0$ (corresponding to $p_1 \leq 0.5$).

Effect for conditional power

For the treatment effect at which the target conditional power should be achieved, either a fixed effect or an interim estimate can be used. The usage of a fixed effect is indicated by setting `useInterimEstimate=FALSE`, in which case the fixed effect is provided by `delta1` on the mean difference scale or by `ncp1` on the non-centrality parameter scale (i.e., `delta1*sqrt(firstStageInformation)`). For an interim estimate, specified by `useInterimEstimate=TRUE`, a lower cut-off for the interim estimate must be provided, either by `delta1Min` on the mean difference scale, or `ncp1Min` on the non-centrality parameter scale. In addition, an upper limit of the estimate may be analogously provided by `delta1Max` or `ncp1Max`.

Sample size and information

The first-stage information of the trial design must be specified to allow for calculations between the mean difference and non-centrality parameter scale. It is provided to the design object via `firstStageInformation`.
Listed below are some examples for the calculation between information (I_1) and sample size:

- One-sample z-test with n total patients: $I_1 = \frac{n}{\sigma^2}$, where σ^2 is the variance of an individual observation
- Balanced two-sample z-test with n_1 patients per group: $I_1 = \frac{1}{2} \cdot \frac{n_1}{\sigma^2}$, where σ^2 is the common variance
- General two-sample z-test with n_1, n_2 patients per group: $I_1 = 1/(\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2})$, where σ_1^2, σ_2^2 are the group-wise variances

Monotonicity

By default, the optimal conditional error function returned by `getDesignOptimalConditionalErrorFunction()` is transformed to be non-increasing in the first-stage p-value p_1 if found to be increasing on any interval. The necessary intervals and constants for the transformation are calculated by `getMonotonisationConstants()`. Although not recommended for the operating characteristics of the design, the transformation may be omitted by setting `enforceMonotonicity=FALSE`.

Constraints

In some applications, it may be feasible to restrict the optimal conditional error function by a lower and/or upper limit. These constraints can be directly implemented on the function by using the arguments `minimumConditionalError` and `maximumConditionalError`. By default, `minimumConditionalError=0` and `maximumConditionalError=1`, i.e., no constraints are applied. The constraints may also be specified on the second-stage information via `minimumSecondStageInformation` and `maximumSecondStageInformation`. If both `minimumConditionalError` and `maximumSecondStageInformation` respectively `maximumConditionalError` and `minimumSecondStageInformation` are provided, both constraints will be applied.

Level constant

The level constant is determined by the helper function `getLevelConstant()`. It is identified using the `uniroot()` function and by default, the interval between 0 and 10 is searched for the level constant. In specific settings, the level constant may lie outside of this interval. In such cases, the search interval can be changed by altering the parameters `levelConstantMinimum` and `levelConstantMaximum`.
If inappropriate constraints to the optimal conditional error function are provided via `minimumConditionalError` and `maximumConditionalError` or `minimumSecondStageInformation` and `maximumSecondStageInformation`, it may be impossible to find a level constant which exhausts the full alpha level.

Generic functions

The `print()` and `plot()` functions are available for objects of class `TrialDesignOptimalConditionalError`.
For details, see `?print.TrialDesignOptimalConditionalError` and `?plot.TrialDesignOptimalConditionalError`.

References

Brannath, W. & Bauer, P. (2004). Optimal conditional error functions for the control of conditional power. Biometrics. https://www.jstor.org/stable/3695393

Brannath, W., Dreher, M., zur Verth, J., Scharpenberg, M. (2024). Optimal monotone conditional error functions. https://arxiv.org/abs/2402.00814

Examples

```
# Create a single-arm design with fixed parameter for the likelihood ratio
# and a fixed effect for conditional power. 80 patients are observed in the
# first-stage (firstStageInformation = 80 in the one-sample test, variance 1).
# The second-stage information is restricted to be between 40 and 160.
getDesignOptimalConditionalErrorFunction(
  alpha = 0.025, alpha1 = 0.001, alpha0 = 0.5, conditionalPower = 0.9,
  delta1 = 0.25, likelihoodRatioDistribution = "fixed", deltaLR = 0.25,
  firstStageInformation = 80, useInterimEstimate = FALSE,
  minimumSecondStageInformation = 40, maximumSecondStageInformation = 160
)

# Create a design comparing two groups using the maximum likelihood ratio
# and an interim estimate for the effect for conditional power.
# 160 patients per arm are observed in the first stage
# (firstStageInformation = 80 in the balanced two-sample test, variance 1).
getDesignOptimalConditionalErrorFunction(
```

```
  alpha = 0.025, alpha1 = 0.001, alpha0 = 0.5, conditionalPower = 0.9,
  delta1Min = 0.25, likelihoodRatioDistribution = "maxlr",
  firstStageInformation = 80, useInterimEstimate = TRUE
)
```

`getExpectedSecondStageInformation` *Calculate Expected Second-stage Information*

Description

Calculate the expected second-stage information using the optimal conditional error function with specific assumptions.

Usage

```
getExpectedSecondStageInformation(
  design,
  likelihoodRatioDistribution = NULL,
  ...
)
```

Arguments

`design` An object of class `TrialDesignOptimalConditionalError` created by `getDesignOptimalConditional`. Contains all necessary arguments to calculate the optimal conditional error function for the specified case.

`likelihoodRatioDistribution`
The distribution to be used for the effect size of the likelihood ratio in the calculation of the expected second-stage information. Options are `"fixed"`, `"normal"`, `"exp"`, `"unif"`, `"maxlr"` for fixed effect size, normally distributed, exponentially distributed, uniformly distributed prior of the effect size and maximum likelihood ratio, respectively. Each case requires different additional specifications:

- `likelihoodRatioDistribution="fixed"` uses one (or more) fixed effect sizes for the likelihood ratio and requires the parameter `deltaLR` which provides the mean difference under which to calculate the likelihood ratio. If `deltaLR` contains multiple values, they may be weighted using an additional argument `weightsDeltaLR`. Omitting `weightsDeltaLR` automatically leads to equal weighting.
- `likelihoodRatioDistribution="normal"` uses a normal prior for the effect size and requires parameters `deltaLR` and `tauLR` for the mean and standard deviation of the normal distribution (both on mean difference scale).

- `likelihoodRatioDistribution="exp"` uses an exponential prior for the effect size and requires the parameter `kappaLR` which is the mean of the exponential distribution (on the mean difference scale).
- `likelihoodRatioDistribution="unif"` uses a uniform prior for the effect size and requires the specification of `deltaMaxLR`, which is the maximum of the support for the uniform likelihood ratio distribution (on the mean difference scale).
- `likelihoodRatioDistribution="maxlr"` estimates the non-centrality parameter to be used for the likelihood ratio from the data. No additional parameters must be specified.

The default is `likelihoodRatioDistribution=NULL`. In this case, the likelihood ratio distribution under which the expected second-stage information is calculated is taken directly from the design object.

`...` Additional parameters required for the specification of `likelihoodRatioDistribution`.

Details

The expected second-stage information is calculated as:

$$\mathbb{E}(I_2) = \int_{\alpha_1}^{\alpha_0} \frac{\nu(\alpha_2(p_1)) \cdot l(p_1)}{\Delta_1^2} dp_1,$$

where

- α_1, α_0 are the first-stage efficacy and futility boundaries
- $\alpha_2(p_1)$ is the optimal conditional error calculated for p_1
- $l(p_1)$ is the "true" likelihood ratio under which to calculate the expected sample size. This can be different from the likelihood ratio used to calibrate the optimal conditional error function.
- Δ_1 is the assumed treatment effect to power for, expressed as a mean difference. It may depend on the interim data (i.e., p_1) in case `useInterimEstimate = TRUE` was specified for the design object.
- $\nu(\alpha_2(p_1)) = (\Phi^{-1}(1 - \alpha_2(p_1)) + \Phi^{-1}(CP))^2$ is a factor calculated for the specific assumptions about the optimal conditional error function and the target conditional power CP.

Value

Expected second-stage information.

References

Brannath, W. & Bauer, P. (2004). Optimal conditional error functions for the control of conditional power. Biometrics. https://www.jstor.org/stable/3695393

See Also

`getDesignOptimalConditionalErrorFunction()`, `getSecondStageInformation()`

Examples

```
# Get a design
design <- getDesignOptimalConditionalErrorFunction(
alpha = 0.025, alpha1 = 0.001, alpha0 = 0.5, conditionalPower = 0.9,
delta1 = 0.25, likelihoodRatioDistribution = "fixed", deltaLR = 0.25,
firstStageInformation = 80, useInterimEstimate = FALSE,
)
# Calculate expected information under correct specification
getExpectedSecondStageInformation(design)

# Calculate expected information under the null hypothesis
getExpectedSecondStageInformation(
 design = design, likelihoodRatioDistribution = "fixed", deltaLR = 0
)
```

---

getLevelConstant *Get Level Constant for Optimal Conditional Error Function*

Description

Find the constant required such that the conditional error function meets the overall level condition.

Usage

```
getLevelConstant(design)
```

Arguments

design	An object of class TrialDesignOptimalConditionalError created by getDesignOptimalConditional Contains all necessary arguments to calculate the optimal conditional error function for the specified case.

Details

The level condition is defined as:

$$\alpha = \alpha_1 + \int_{\alpha_1}^{\alpha_0} \alpha_2(p_1)dp_1.$$

The constant c_0 of the optimal conditional error function is calibrated such that it meets the level condition. For a valid design, the additional following condition must be met to be able to exhaust the level α:

$$\alpha_1 + CP(\alpha_0 - \alpha_1) > \alpha.$$

This condition is checked by `getLevelConstant()` and the execution is terminated if it is not met. In case a conditional power function is used, the condition is instead:

$$\alpha_1 + \int_{\alpha_1}^{\alpha_0} CP(p_1)dp_1 > \alpha.$$

Value

A list that contains the constant (element `$root`) and other components provided by `uniroot()`. The level constant is calculated corresponding to the mean difference scale.

References

Brannath, W. & Bauer, P. (2004). Optimal conditional error functions for the control of conditional power. Biometrics. https://www.jstor.org/stable/3695393

Brannath, W., Dreher, M., zur Verth, J., Scharpenberg, M. (2024). Optimal monotone conditional error functions. https://arxiv.org/abs/2402.00814

`getLikelihoodRatio` *Calculate Likelihood Ratio*

Description

Calculate the likelihood ratio of a p-value for a given distribution.

Usage

```
getLikelihoodRatio(firstStagePValue, design)
```

Arguments

`firstStagePValue`	First-stage p-value or p-values. Must be a numeric vector between 0 and 1.
`design`	An object of class `TrialDesignOptimalConditionalError` created by `getDesignOptimalConditional` Contains all necessary arguments to calculate the optimal conditional error function for the specified case.

Details

The calculation of the likelihood ratio for a first-stage p-value p_1 is done based on a distributional assumption, specified in the `design` object. The different options require different parameters, elaborated in the following.

- `likelihoodRatioDistribution="fixed"`: calculates the likelihood ratio for a fixed Δ. The non-centrality parameter of the likelihood ratio ϑ is then computed as `deltaLR*sqrt(firstStageInformation)` and the likelihood ratio is calculated as:

$$l(p_1) = e^{\Phi^{-1}(1-p_1)\vartheta - \vartheta^2/2}.$$

`deltaLR` may also contain multiple elements, in which case a weighted likelihood ratio is calculated for the given values. Unless positive weights that sum to 1 are provided by the argument `weightsDeltaLR`, equal weights are assumed.

- `likelihoodRatioDistribution="normal"`: calculates the likelihood ratio for a normally distributed prior of ϑ with mean `deltaLR*sqrt(firstStageInformation)` (μ) and standard deviation `tauLR*sqrt(firstStageInformation)` (σ). The parameters `deltaLR` and `tauLR` must be specified on the mean difference scale.

$$l(p_1) = (1+\sigma^2)^{-\frac{1}{2}} \cdot e^{-(\mu/\sigma)^2/2+(\sigma\Phi^{-1}(1-p_1)+\mu/\sigma)^2/(2\cdot(1+\sigma^2))}$$

- `likelihoodRatioDistribution="exp"`: calculates the likelihood ratio for an exponentially distributed prior of ϑ with mean `kappaLR*sqrt(firstStageInformation)` (η). The likelihood ratio is then calculated as:

$$l(p_1) = \eta \cdot \sqrt{2\pi} \cdot e^{(\Phi^{-1}(1-p_1)-\eta)^2/2} \cdot \Phi(\Phi^{-1}(1-p_1)-\eta)$$

- `likelihoodRatioDistribution="unif"`: calculates the likelihood ratio for a uniformly distributed prior of ϑ on the support $[0, \Delta \cdot \sqrt{I_1}]$, where Δ is specified as `deltaMaxLR` and I_1 is the `firstStageInformation`.

$$l(p_1) = \frac{\sqrt{2\pi}}{\Delta \cdot \sqrt{I_1}} \cdot e^{\Phi^{-1}(1-p_1)^2/2} \cdot (\Phi(\Delta \cdot \sqrt{I_1} - \Phi^{-1}(1-p_1)) - p_1)$$

- `likelihoodRatioDistribution="maxlr"`: the non-centrality parameter ϑ is estimated from the data and no additional parameters must be specified. The likelihood ratio is estimated from the data as:

$$l(p_1) = e^{max(0,\Phi^{-1}(1-p_1))^2/2}$$

  The maximum likelihood ratio is always restricted to effect sizes $\vartheta \geq 0$ (corresponding to $p_1 \leq 0.5$).

Value

The value of the likelihood ratio for the given specification.

References

Brannath, W. & Bauer, P. (2004). Optimal conditional error functions for the control of conditional power. Biometrics. https://www.jstor.org/stable/3695393

Hung, H. M. J., O'Neill, R. T., Bauer, P. & Kohne, K. (1997). The behavior of the p-value when the alternative hypothesis is true. Biometrics. https://www.jstor.org/stable/2533093

Examples

```
# Get a design
design <- getDesignOptimalConditionalErrorFunction(
alpha = 0.025, alpha1 = 0.001, alpha0 = 0.5, conditionalPower = 0.9,
delta1 = 0.25, likelihoodRatioDistribution = "fixed", deltaLR = 0.25,
firstStageInformation = 80, useInterimEstimate = FALSE,
)

getLikelihoodRatio(firstStagePValue = c(0.05, 0.1, 0.2), design = design)
```

`getMonotoneFunction` *Return Monotone Function Values*

Description

Applies the provided monotonisation constants to a specified, possibly non-monotone function. The returned function values are non-increasing.

Usage

```
getMonotoneFunction(
  x,
  fun,
  lower = NULL,
  upper = NULL,
  argument = NULL,
  nSteps = 10^4,
  epsilon = 10^(-5),
  numberOfIterationsQ = 10^4,
  design
)
```

Arguments

`x`	Argument values.
`fun`	The function to be made monotone.
`lower`	The lower limit of the interval on which the function should be monotonised. Must be a numeric value.
`upper`	The upper limit of the interval on which the function should be monotonised.
`argument`	The argument in which the function should be monotonised, given as a character.
`nSteps`	The number of steps to be taken when checking the function for monotonicity. Must be a numeric value. Default 10^4.
`epsilon`	Maximum allowed difference between the initial and monotone integral. Must be a numeric value. Default 10^-5.
`numberOfIterationsQ`	Maximum number of iterations allowed to determine each value of q. Must be a numeric value. Default 10^4.
`design`	An object of class `TrialDesignOptimalConditionalError` created by `getDesignOptimalConditional` Contains all necessary arguments to calculate the optimal conditional error function for the specified case.

Details

The exact monotonisation process is outlined in Brannath et al. (2024), but specified in terms of the first-stage test statistic z_1 rather than the first-stage p-value p_1.
The algorithm can easily be translated to the use of p-values by switching the maximum and minimum functions, i.e., replacing $\min\{q, Q(z_1)\}$ by $\max\{q, Q(p_1)\}$ and $\min\{q, Q(z_1)\}$ by $\max\{q, Q(p_1\}$.

Value

Monotone function values.

References

Brannath, W., Dreher, M., zur Verth, J., Scharpenberg, M. (2024). Optimal monotone conditional error functions. https://arxiv.org/abs/2402.00814

`getMonotonisationConstants`
Calculate the Constants for Monotonisation

Description

Computes the constants required to make a function non-increasing on the specified interval. The output of this function is necessary to calculate the monotone optimal conditional error function. The output object is a list that contains the intervals on which constant values are required, specified by the minimum `dls` and maximum `dus` of the interval and the respective constants, `qs`.

Usage

```
getMonotonisationConstants(
  fun,
  lower = 0,
  upper = 1,
  argument,
  nSteps = 10^4,
  epsilon = 10^(-5),
  numberOfIterationsQ = 10^4,
  design
)
```

Arguments

`fun`	The function to be made monotone.
`lower`	The lower limit of the interval on which the function should be monotonised. Must be a numeric value.
`upper`	The upper limit of the interval on which the function should be monotonised.
`argument`	The argument in which the function should be monotonised, given as a character.
`nSteps`	The number of steps to be taken when checking the function for monotonicity. Must be a numeric value. Default 10^4.
`epsilon`	Maximum allowed difference between the initial and monotone integral. Must be a numeric value. Default 10^-5.
`numberOfIterationsQ`	Maximum number of iterations allowed to determine each value of q. Must be a numeric value. Default 10^4.

`design` An object of class `TrialDesignOptimalConditionalError` created by `getDesignOptimalConditional`. Contains all necessary arguments to calculate the optimal conditional error function for the specified case.

Value

A list containing the monotonisation constants (element `$qs`) and the intervals on which they must be applied, specified via minimum (element `qls`) and maximum (element `qus`).

References

Brannath, W., Dreher, M., zur Verth, J., Scharpenberg, M. (2024). Optimal monotone conditional error functions. https://arxiv.org/abs/2402.00814

---

getNu *Calculate Nu*

Description

Calculate the factor which relates α_2 to the second-stage information for given conditional power.

Usage

```
getNu(alpha, conditionalPower)
```

Arguments

`alpha` The (conditional) type I error rate of the design. Must be a numeric vector with values between 0 and 1.

`conditionalPower` The target conditional power CP of the design. Must be a numeric value.

Details

Note that this function uses factor 1 instead of factor 2 (Brannath & Bauer 2004). This has no impact on the optimal conditional error function, as constant factors are absorbed by the level constant c_0. The calculation is:

$$\nu(\alpha_2(p_1)) = (\Phi^{-1}(1-\alpha_2(p_1)) + \Phi^{-1}(CP))^2.$$

Value

Factor linking information and α_2.

References

Brannath, W. & Bauer, P. (2004). Optimal conditional error functions for the control of conditional power. Biometrics. https://www.jstor.org/stable/3695393

Examples

```
getNu(alpha = 0.05, conditionalPower = 0.9)

# Returns 0 if alpha exceeds conditionalPower
getNu(alpha = 0.8, conditionalPower = 0.7)
```

`getNuPrime` *Calculate the Derivate of Nu*

Description

Calculates the derivative of nu for a given conditional error and conditional power.

Usage

```
getNuPrime(alpha, conditionalPower)
```

Arguments

`alpha`	The (conditional) type I error rate of the design. Must be a numeric vector with values between 0 and 1.
`conditionalPower`	The target conditional power CP of the design. Must be a numeric value.

Details

The function ν' is defined as

$$\nu'(p_1) = -2 \cdot (\Phi^{-1}(1 - \alpha_2(p_1)) + \Phi^{-1}(CP))/\phi(\Phi^{-1}(1 - \alpha_2(p_1))).$$

Note that in this implementation, the the factor -2 is used instead of -4, which is used in by Brannath & Bauer (2004), who explicitly investigate the setting of a balanced two-group trial. The argument `conditionalPower` is either the fixed target conditional power or the value of the conditional power function at the corresponding first-stage p-value.

Value

Value for nu prime.

References

Brannath, W. & Bauer, P. (2004). Optimal conditional error functions for the control of conditional power. Biometrics. https://www.jstor.org/stable/3695393

Examples

```
getNuPrime(alpha = 0.05, conditionalPower = 0.9)
```

`getOptimalConditionalError` *Calculate the Optimal Conditional Error*

Description

Calculate the Optimal Conditional Error

Usage

```
getOptimalConditionalError(firstStagePValue, design)
```

Arguments

`firstStagePValue`	First-stage p-value or p-values. Must be a numeric vector between 0 and 1.
`design`	An object of class `TrialDesignOptimalConditionalError` created by `getDesignOptimalConditional` Contains all necessary arguments to calculate the optimal conditional error function for the specified case.

Details

The optimal conditional error α_2 given a first-stage p-value p_1 is calculated as:

$$\alpha_2(p_1) = \psi(-e^{c_0} \cdot \frac{\Delta_1^2}{l(p_1)}).$$

The level constant c_0 as well as the specification of the effect size Δ_1 and the likelihood ratio $l(p_1)$ must be contained in the `design` object (see `?getDesignOptimalConditionalErrorFunction`). Early stopping rules are supported, i.e., for $p_1 \leq \alpha_1$, the returned conditional error is 1 and for $p_1 > \alpha_0$, the returned conditional error is 0.

Value

Value of the optimal conditional error function.

References

Brannath, W. & Bauer, P. (2004). Optimal conditional error functions for the control of conditional power. Biometrics. https://www.jstor.org/stable/3695393

See Also

`getDesignOptimalConditionalErrorFunction()`

Examples

```
# Create a design
design <- getDesignOptimalConditionalErrorFunction(
alpha = 0.025, alpha1 = 0.001, alpha0 = 0.5, conditionalPower = 0.9,
delta1 = 0.5, firstStageInformation = 40, useInterimEstimate = FALSE,
likelihoodRatioDistribution = "fixed", deltaLR = 0.5)

# Calculate optimal conditional error
getOptimalConditionalError(
firstStagePValue = c(0.1, 0.2, 0.3), design = design
)
```

---

getOverallPower *Calculate the overall power*

Description

Calculate the overall power and other operating characteristics of a design.

Usage

```
getOverallPower(design, alternative)
```

Arguments

design	An object of class TrialDesignOptimalConditionalError created by getDesignOptimalConditional Contains all necessary arguments to calculate the optimal conditional error function for the specified case.
alternative	Assumed relative effect size.

Details

This function is used to evaluate the overall performance of a design. The probabilities for first-stage futility, first-stage efficacy and overall efficacy (i.e., overall power) are saved in an object of class PowerResultsOptimalConditionalError.

Value

The overall power of the design at the provided effect size.

See Also

getDesignOptimalConditionalErrorFunction(), getSimulationResults()

Examples

```
# Get a design
design <- getDesignOptimalConditionalErrorFunction(
alpha = 0.025, alpha1 = 0.001, alpha0 = 0.5, conditionalPower = 0.9,
delta1 = 0.25, likelihoodRatioDistribution = "fixed", deltaLR = 0.25,
firstStageInformation = 80, useInterimEstimate = FALSE,
)

getOverallPower(design, alternative = 0.25)
```

`getPsi` *Calculate Psi, the Inverse of Nu Prime*

Description

Get point-wise values of psi (inverse of nu prime)

Usage

```
getPsi(nuPrime, conditionalPower, design, firstStagePValue)
```

Arguments

`nuPrime`	The function value to be inverted.
`conditionalPower`	The target conditional power CP of the design. Must be a numeric value.
`design`	An object of class `TrialDesignOptimalConditionalError` created by `getDesignOptimalConditional` Contains all necessary arguments to calculate the optimal conditional error function for the specified case.
`firstStagePValue`	First-stage p-value or p-values. Must be a numeric vector between 0 and 1.

Details

The function ψ is the inverse of:

$$\nu'(\alpha) = -2 \cdot (\Phi^{-1}(1-\alpha) + \Phi^{-1}(1-CP))/\phi(\Phi^{-1}(1-\alpha))$$

. If the conditional power CP lies outside of the range $1 - \Phi(2) \leq CP \leq \Phi(2)$, the calculation is slightly more complicated (design and firstStagePValue need to be provided). The argument `conditionalPower` is either the fixed target conditional power or the value of the conditional power function at the corresponding first-stage p-value.

Value

The value of alpha which corresponds to nuPrime and lies between 0 and `conditionalPower`.

Examples

```
# Returns 0.05
getPsi(getNuPrime(alpha = 0.05, conditionalPower = 0.9), conditionalPower = 0.9,
  design = NULL, firstStagePValue = NULL)
```

`getQ` *Calculate Q*

Description

Calculate the ratio of likelihood ratio and squared effect size.

Usage

```
getQ(firstStagePValue, design)
```

Arguments

`firstStagePValue`	First-stage p-value or p-values. Must be a numeric vector between 0 and 1.
`design`	An object of class `TrialDesignOptimalConditionalError` created by `getDesignOptimalConditional` Contains all necessary arguments to calculate the optimal conditional error function for the specified case.

Details

For more information on how to specify the likelihood ratio, see `?getLikelihoodRatio()`. In case the optimal conditional error function is ever increasing in the first-stage p-value p_1, a monotone transformation of `getQ()` is needed for logical consistency and type I error rate control.
The formula for $Q(p_1)$ is:

$$Q(p_1) = l(p_1)/\Delta_1^2,$$

where $l(p_1)$ is the likelihood ratio and Δ_1 is the effect size at which the conditional power should be achieved. The effect size may also depend on the interim data (i.e., on p_1) in case `useInterimEstimate = TRUE` was specified for the design object.

Value

Ratio of likelihood ratio and squared effect size.

References

Brannath, W., Dreher, M., zur Verth, J., Scharpenberg, M. (2024). Optimal monotone conditional error functions. https://arxiv.org/abs/2402.00814

Examples

```
# Get a design
design <- getDesignOptimalConditionalErrorFunction(
alpha = 0.025, alpha1 = 0.001, alpha0 = 0.5, conditionalPower = 0.9,
delta1 = 0.25, likelihoodRatioDistribution = "fixed", deltaLR = 0.25,
firstStageInformation = 80, useInterimEstimate = FALSE,
)

getQ(firstStagePValue = c(0.05, 0.1, 0.2), design = design)
```

---

getSecondStageInformation

Calculate the Second-stage Information

Description

Calculate second-stage information for given first-stage p-value and design.

Usage

```
getSecondStageInformation(firstStagePValue, design)
```

Arguments

`firstStagePValue`	First-stage p-value or p-values. Must be a numeric vector between 0 and 1.
`design`	An object of class `TrialDesignOptimalConditionalError` created by `getDesignOptimalConditional` Contains all necessary arguments to calculate the optimal conditional error function for the specified case.

Details

The second-stage information I_2 is calculated given a first-stage p-value p_1 as:

$$I_2(p_1) = \frac{(\Phi^{-1}(1-\alpha_2(p_1)) + \Phi^{-1}(CP))^2}{\Delta_1^2} = \frac{\nu(\alpha_2(p_1))}{\Delta_1^2},$$

where

- $\alpha_2(p_1)$ is the conditional error function
- CP is the target conditional power
- Δ_1 is the assumed treatment effect (expressed as a mean difference).

The conditional error is calculated according to the specification provided in the `design` argument. For p-values smaller or equal to the first-stage efficacy boundary as well as p-values greater than the first-stage futility boundary, the returned information is 0 (since the trial is ended early in both cases).

Value

The second-stage information.

References

Brannath, W. & Bauer, P. (2004). Optimal conditional error functions for the control of conditional power. Biometrics. https://www.jstor.org/stable/3695393

See Also

getDesignOptimalConditionalErrorFunction(), getExpectedSecondStageInformation(), getOptimalConditional

Examples

```
design <- getDesignOptimalConditionalErrorFunction(
  alpha = 0.025, alpha1 = 0.001, alpha0 = 0.5,
  conditionalPower = 0.9, delta1 = 0.25, useInterimEstimate = FALSE,
  firstStageInformation = 40, likelihoodRatioDistribution = "maxlr"
)

getSecondStageInformation(
  firstStagePValue = c(0.05, 0.1, 0.2), design = design
)
```

---

getSimulationResults *Simulate trials*

Description

Simulate the rejection probability for a given design and alternative.

Usage

```
getSimulationResults(
  design,
  maxNumberOfIterations = 10000,
  alternative,
  seed = NULL
)
```

Arguments

design	An object of class TrialDesignOptimalConditionalError created by getDesignOptimalConditional Contains all necessary arguments to calculate the optimal conditional error function for the specified case.
maxNumberOfIterations	Number of trials to be simulated.

alternative	Assumed relative effect size.
seed	An optional seed for reproducibility.

Details

Simulates the probabilities of overall rejection as well as early futility and early efficacy for the provided scenario and design. This is done by generating random normally distributed test statistics and calculating their p-values.

Value

An object of class SimulationResultsOptimalConditionalError containing the simulation results.

See Also

getDesignOptimalConditionalErrorFunction(), getOverallPower()

Examples

```
design <- getDesignOptimalConditionalErrorFunction(
 alpha = 0.025, alpha1 = 0.001, alpha0 = 0.5, delta1 = 0.25,
 useInterimEstimate = FALSE,
 conditionalPower = 0.9, likelihoodRatioDistribution = "maxlr",
 firstStageInformation = 10
)

# Simulate under the null hypothesis and for a mean difference of 0.5
getSimulationResults(
 design = design, alternative = c(0, 0.5)
)
```

---

plot.TrialDesignOptimalConditionalError

Plot the optimal conditional error function

Description

The returned plot is a ggplot2 object and can be supplemented with additional layers using ggplot2 commands.

Usage

```
## S3 method for class 'TrialDesignOptimalConditionalError'
plot(x, range = c(0, 1), type = 1, plotNonMonotoneFunction = FALSE, ...)
```

Arguments

x	Design object of class TrialDesignOptimalConditionalError.
range	Numeric vector with two entries specifying the range of the x-axis of the plot.
type	Type of plot to be created. Options are: • type = 1: Plot the values of the optimal conditional error function against the first-stage p-value. • type = 2: Plot the second-stage information resulting from the optimal conditional error function against the first-stage p-value. • type = 3: Plot the likelihood ratio of the given specification of the optimal conditional error function against the first-stage p-value. • type = 4: Plot the function Q of the given specification of the optimal conditional error function against the first-stage p-value.
plotNonMonotoneFunction	Logical. Should the non-monotone version of the plot be drawn? Not applicable for plot type 3. Default: FALSE.
...	Additional arguments required for generic compatibility

Value

No return value, plots the design.

PowerResultsOptimalConditionalError *Power results for optimal conditional error design*

Description

A class for power results of the optimal conditional error function.

---

print.PowerResultsOptimalConditionalError *Print power results*

Description

Print an overview of exact power results.

Usage

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

Arguments

x	Power results object of class PowerResultsOptimalConditionalError
...	Additional arguments required for generic compatibility

Value

No return value, prints the power calculation results.

---

print.SimulationResultsOptimalConditionalError
Print simulation results

Description

Print an overview of simulation results.

Usage

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

Arguments

x	Simulation results object of class SimulationResultsOptimalConditionalError
...	Additional arguments required for generic compatibility

Value

No return value, prints the simulation results.

---

print.TrialDesignOptimalConditionalError
Print optimal conditional error trial design

Description

Print an overview of the specified design parameters.

Usage

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

Arguments

x	Design object of class TrialDesignOptimalConditionalError
...	Additional arguments required for generic compatibility

Value

No return value, prints the design.

SimulationResultsOptimalConditionalError

Simulation results for optimal conditional error design

Description

A class for simulation results of the optimal conditional error function.

---

summary.TrialDesignOptimalConditionalError

Summary of the optimal conditional error trial design

Description

Provide an overview of the operating characteristics of the optimal conditional error trial design.

Usage

```
## S3 method for class 'TrialDesignOptimalConditionalError'
summary(object, ...)
```

Arguments

object	Design object of class TrialDesignOptimalConditionalError
...	Additional arguments required for generic compatibility

Value

No return value, prints a summary of design performance.

`TrialDesignOptimalConditionalError` *Optimal Conditional Error Design*

Description

A class for a trial design object using the optimal conditional error function.

Details

This object should not be created directly; use `getDesignOptimalConditionalErrorFunction()` with suitable arguments to create a design.

Index

.rangeCheck, 2

getConstraintC_max, 3
getConstraintC_min, 3
getDesignOptimalConditionalErrorFunction, 4
getDesignOptimalConditionalErrorFunction(), *11*, *19*, *20*, *24*, *25*
getExpectedSecondStageInformation, 10
getExpectedSecondStageInformation(), *24*
getLevelConstant, 12
getLikelihoodRatio, 13
getMonotoneFunction, 15
getMonotonisationConstants, 16
getNu, 17
getNuPrime, 18
getOptimalConditionalError, 19
getOptimalConditionalError(), *24*
getOverallPower, 20
getOverallPower(), *25*
getPsi, 21
getQ, 22
getSecondStageInformation, 23
getSecondStageInformation(), *11*
getSimulationResults, 24
getSimulationResults(), *20*

plot.TrialDesignOptimalConditionalError, 25
PowerResultsOptimalConditionalError, 26
print.PowerResultsOptimalConditionalError, 26
print.SimulationResultsOptimalConditionalError, 27
print.TrialDesignOptimalConditionalError, 27

SimulationResultsOptimalConditionalError, 28
summary.TrialDesignOptimalConditionalError, 28

TrialDesignOptimalConditionalError, 29