Package 'rpact'

October 15, 2018

Title R Package for Adaptive Clinical Trials

Version 1.0.0

Date 2018-10-11

Description Design and Analysis of Confirmatory Adaptive Clinical Trials with Continuous, Binary, and Survival Endpoint.

Depends R (>= 3.3.0)

License GPL-3

Encoding UTF-8

LazyData true

URL https://www.rpact.org

BugReports https://bugreport.rpact.org

Imports methods,
stats,
utils,
graphics

Suggests parallel,
ggplot2 (>= 2.2.0),
testthat (>= 2.0.0)

RoxygenNote 6.0.1

Collate 'f_core_constants.R'
'class_core_parameter_set.R'
'class_core_plot_settings.R'
'class_analysis_dataset.R'
'class_design.R'
'class_analysis_stage_results.R'
'class_analysis_results.R'
'class_design_plan.R'
'class_design_power_and_asn.R'
'class_design_set.R'
'f_analysis.R'
'f_analysis_means.R'
'f_analysis_rates.R'
'f_analysis_survival.R'
'f_core_assertions.R'
'f_core_output_formats.R'

'f_core_utilities.R'
'f_design_fisher_combination_test.R'
'f_design_group_sequential.R'
'f_design_sample_size_calculator.R'
'pkgname.R'

R topics documented:

`rpact-package` *RPACT - R Package for Adaptive Clinical Trials*

Description

RPACT is a comprehensive package that enables the design and analysis of confirmatory adaptive group sequential designs. Particularly, the methods described in the recent monograph by Wassmer and Brannath (published by Springer, 2016) are implemented.

Details

RPACT includes the classical group sequential designs (incl user spending function approaches) where the sample sizes per stage (or the time points of interim analysis) cannot be changed in a data-driven way. Confirmatory adaptive designs explicitly allow for this under control of the Type I error rate. They are either based on the combination testing or the conditional rejection probability

(CRP) principle. Both are available, for the former the inverse normal combination test and Fisher's combination test can be used.

Specific results of the adaptive methodology are also available, e.g., overall confidence intervals and p-values and conditional and predictive power assessments.

Designs are available for trials with continuous, binary, and survival endpoint.

For more information please visit www.rpact.org. If you are interested in professional services round about the package or need a comprehensive validation documentation to fulfill regulatory requirements please visit www.rpact.com.

RPACT is developed by

- Gernot Wassmer (gernot.wassmer@rpact.com) and
- Friedrich Pahlke (friedrich.pahlke@rpact.com).

Author(s)

Maintainer: Friedrich Pahlke `<friedrich.pahlke@rpact.com>`

Authors:

- Gernot Wassmer `<gernot.wassmer@rpact.com>`

References

Wassmer, G., Brannath, W. (2016) Group Sequential and Confirmatory Adaptive Designs in Clinical Trials (Springer Series in Pharmaceutical Statistics)

See Also

Useful links:

- `https://www.rpact.org`
- Report bugs at `https://bugreport.rpact.org`

`AnalysisResults` *Basic Class for Analysis Results*

Description

A basic class for analysis results.

Details

`AnalysisResults` is the basic class for

- `AnalysisResultsFisher`,
- `AnalysisResultsGroupSequential`, and
- `AnalysisResultsInverseNormal`.

AnalysisResultsFisher
Analysis Results Fisher

Description

Class for analysis results based on a Fisher design.

Details

This object can not be created directly; use `getAnalysisResults` with suitable arguments to create the analysis results of a Fisher design.

AnalysisResultsGroupSequential
Analysis Results Group Sequential

Description

Class for analysis results results based on a group sequential design.

Details

This object can not be created directly; use `getAnalysisResults` with suitable arguments to create the analysis results of a group sequential design.

AnalysisResultsInverseNormal
Analysis Results Inverse Normal

Description

Class for analysis results results based on an inverse normal design.

Details

This object can not be created directly; use `getAnalysisResults` with suitable arguments to create the analysis results of a inverse normal design.

AnalysisResults_as.data.frame
Coerce AnalysisResults to a Data Frame

Description

Returns the `AnalysisResults` object as data frame.

Usage

```
## S3 method for class 'AnalysisResults'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)
```

Details

Coerces the analysis results to a data frame.

AnalysisResults_names
The Names of a Analysis Results object

Description

Function to get the names of a `AnalysisResults` object.

Usage

```
## S3 method for class 'AnalysisResults'
names(x)
```

Details

Returns the names of a analysis results that can be accessed by the user.

AnalysisResults_plot
Analysis Results Plotting - Conditional Power Plot

Description

Plots the conditional power together with the likelihood function.

Usage

```
## S3 method for class 'AnalysisResults'
plot(x, y, ..., nPlanned = NA_real_,
  stage = x$getNumberOfStages(), allocationRatioPlanned = NA_real_,
  main = NA_character_, xlab = NA_character_, ylab = NA_character_,
  legendTitle = "", palette = "Set1", legendPosition = NA_integer_,
  type = 1)
```

Arguments

x	The analysis results at given stage, obtained from getAnalysisResults.
y	Not available for this kind of plot (is only defined to be compatible to the generic plot function).
...	Optional ggplot2 arguments.
nPlanned	The sample size planned for the subsequent stages. It should be a vector with length equal to the remaining stages and is the overall sample size in the two treatment groups if two groups are considered.
stage	The stage number (optional). Default: total number of existing stages in the data input used to create the analysis results.
allocationRatioPlanned	The allocation ratio n1/n2 for two treatment groups planned for the subsequent stages, the default value is 1.
main	The main title, default is "Dataset".
xlab	The x-axis label, default is "Stage".
ylab	The y-axis label.
legendTitle	The legend title, default is "".
palette	The palette, default is "Set1".
legendPosition	The position of the legend. By default (NA_integer_) the algorithm tries to find a suitable position. Choose one of the following values to specify the position manually: • 0: legend position outside plot • 1: legend position left top • 2: legend position left center • 3: legend position left bottom • 4: legend position right top • 5: legend position right center • 6: legend position right bottom
type	The plot type (default = 1). Note that at the moment only one type (the conditional power plot) is available.

Details

The conditional power is calculated only if effect size and sample size is specified.

Examples

```
design <- getDesignGroupSequential(kMax = 4, alpha = 0.025,
    informationRates = c(0.2, 0.5, 0.8, 1),
    typeOfDesign = "WT", deltaWT = 0.25)

dataExample <- getDataset(
    n = c(20, 30, 30),
    means = c(0.5, 0.51, 0.55)*100,
    stDevs = c(1.3, 1.4, 1.2)*100
)

result <- getAnalysisResults(design = design,
    dataInput = dataExample, thetaH0 = 20,
    nPlanned = c(30), thetaH1 = 1.5)

if (require(ggplot2)) plot(result, thetaRange = c(0, 100))
```

Dataset *Dataset*

Description

Basic class for datasets.

Details

`Dataset` is the basic class for

- `DatasetMeans`,
- `DatasetRates`, and
- `DatasetSurvival`.

This basic class contains the fields `stages` and `groups` and several commonly used functions.

Fields

`stages` The stage numbers.

`groups` The group numbers.

DatasetMeans *Dataset of Means*

Description

Class for a dataset of means.

Details

This object can not be created directly; better use `getDataset` with suitable arguments to create a dataset of means.

Fields

`groups` The group numbers.

`stages` The stage numbers.

`sampleSizes` The sample sizes.

`means` The means.

`stDevs` The standard deviations.

DatasetRates *Dataset of Rates*

Description

Class for a dataset of rates.

Details

This object can not be created directly; better use `getDataset` with suitable arguments to create a dataset of rates.

Fields

`group` The group numbers.

`stage` The stage numbers.

`sampleSize` The sample sizes.

`event` The events.

DatasetSurvival *Dataset of Survival Data*

Description

Class for a dataset of survival data.

Details

This object can not be created directly; better use `getDataset` with suitable arguments to create a dataset of survival data.

Fields

`group` The group numbers.

`stage` The stage numbers.

`overallEvent` The overall events.

`overallAllocationRatio` The overall allocations ratios.

`overallLogRank` The overall logrank test statistics.

Dataset_plot *Dataset Plotting*

Description

Plots a dataset.

Usage

```
## S3 method for class 'Dataset'
plot(x, y, ..., main = "Dataset", xlab = "Stage",
  ylab = NA_character_, legendTitle = "Group", palette = "Set1")
```

Arguments

`x`	The `Dataset` object to plot.
`y`	Not available for this kind of plot (is only defined to be compatible to the generic plot function).
`...`	Optional `ggplot2` arguments.
`main`	The main title, default is `"Dataset"`.
`xlab`	The x-axis label, default is `"Stage"`.
`ylab`	The y-axis label.
`legendTitle`	The legend title, default is `"Group"`.
`palette`	The palette, default is `"Set1"`.

Details

Generic function to plot all kinds of datasets.

Examples

```
# Plot a dataset of means
dataExample <- getDataset(
    n1 = c(22, 11, 22, 11),
    n2 = c(22, 13, 22, 13),
    means1 = c(1, 1.1, 1, 1),
    means2 = c(1.4, 1.5, 3, 2.5),
    stDevs1 = c(1, 2, 2, 1.3),
    stDevs2 = c(1, 2, 2, 1.3))

if (require(ggplot2)) plot(dataExample, main = "Comparison of means")

# Plot a dataset of rates
dataExample <- getDataset(
    n1 = c(8, 10, 9, 11),
    n2 = c(11, 13, 12, 13),
    events1 = c(3, 5, 5, 6),
    events2 = c(8, 10, 12, 12)
)

if (require(ggplot2)) plot(dataExample, main = "Comparison of rates")
```

FieldSet *Field Set*

Description

Basic class for field sets.

Details

The field set implements basic functions for a set of fields.

FieldSet_names *The Names of a Field Set object*

Description

Function to get the names of a `FieldSet` object.

Usage

```
## S3 method for class 'FieldSet'
names(x)
```

Details

Returns the names of a field set that can be accessed by the user.

FieldSet_print *Print Field Set Values*

Description

`print` prints its `FieldSet` argument and returns it invisibly (via `invisible(x)`).

Usage

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

Details

Prints the field set.

---

getAnalysisResults *Get Analysis Results*

Description

Calculates and returns the analysis results for the specified design and data.

Usage

```
getAnalysisResults(design, dataInput, ...,
  directionUpper = C_DIRECTION_UPPER_DEFAULT, thetaH0 = NA_real_,
  nPlanned = NA_real_)
```

Arguments

`design`	The trial design.
`dataInput`	The summary data used for calculating the test results. This is either an element of `DatasetMeans`, of `DatasetRates`, or of `DatasetSurvival`. For more information see `details` below.
`...`	Further arguments to be passed to methods (cp. separate functions in See Also), e.g.,

stage The stage number (optional). Default: total number of existing stages in the data input.

allocationRatioPlanned The allocation ratio n1/n2 for two treatment groups planned for the subsequent stages, the default value is 1.

thetaH1 and assumedStDev or pi1, pi2 The assumed effect size or assumed rates to calculate the conditional power. Depending on the type of dataset, either thetaH1 (means and survival) or pi1, pi2 (rates) can be specified. Additionally, if testing means is specified, an assumed standard deviation can be specified, default is 1.

normalApproximation The type of computation of the p-values. Default is FALSE for testing means (i.e., the t test is used) and TRUE for testing rates and the hazard ratio. For testing rates, if `normalApproximation = FALSE` is specified, the binomial test (one sample) or the test of Fisher (two samples) is used for calculating the p-values. In the survival setting, `normalApproximation = FALSE` has no effect.

equalVariances The type of t test. For testing means in two treatment groups, either the t test assuming that the variances are equal or the t test without assuming this, i.e., the test of Welch-Satterthwaite is calculated, default is `equalVariances = TRUE`.

iterations Iterations for simulating the power for Fisher's combination test. If the power for more than one remaining stages is to be determined for Fisher's combination test, it is estimated via simulation with specified `iterations`, the default value is 10000.

seed Seed for simulating the power for Fisher's combination test. See above, default is a random seed.

`directionUpper`
: The direction of one-sided testing. Default is `directionUpper = TRUE` which means that larger values of the test statistics yield smaller p-values.

`thetaH0`
: The null hypothesis value, default is 0 for the normal and the binary case, it is 1 for the survival case. For testing a rate in one sample, a value thetaH0 in (0, 1) has to be specified for defining the null hypothesis H0: pi = thetaH0.
For noninferiority designs, this is the noninferiority bound.

`nPlanned`
: The sample size planned for the subsequent stages. It should be a vector with length equal to the remaining stages and is the overall sample size in the two treatment groups if two groups are considered.

Details

Given a design and a dataset, at given stage the function calculates the test results (effect sizes, stage-wise test statistics and p-values, overall p-values and test statistics, conditional rejection probability (CRP), conditional power, Repeated Confidence Intervals (RCIs), repeated overall p-values, and final stage p-values, median unbiased effect estimates, and final confidence intervals.

`dataInput` is either an element of `DatasetMeans`, of `DatasetRates`, or of `DatasetSurvival` and should be created with the function `getDataset`.

Value

Returns an `AnalysisResults` object.

Note

The conditional power is calculated only if effect size and sample size is specified. Median unbiased effect estimates and confidence intervals are calculated if a group sequential design or an inverse normal combination test design was chosen, i.e., it is not applicable for Fisher's p-value combination test design.

A final stage p-value for Fisher's combination test is calculated only if a two-stage design was chosen. For Fisher's combination test, the conditional power for more than one remaining stages is estimated via simulation.

See Also

Alternatively the analysis results can be calculated separately using one of the following functions:

- getTestActions,
- getConditionalPower,
- getConditionalRejectionProbabilities,
- getRepeatedConfidenceIntervals,
- getRepeatedPValues,
- getFinalConfidenceInterval,
- getFinalPValue.

Examples

```
design <- getDesignGroupSequential()
dataMeans <- getDataset(
    n = c(10,10),
    means = c(1.96,1.76),
    stDevs = c(1.92,2.01))

getAnalysisResults(design, dataMeans)

# produces:
#
# Analysis results (group sequential design):
#  Stages                          : 1, 2, 3
#  Information rates               : 0.333, 0.667, 1.000
#  Critical values                 : 3.471, 2.454, 2.004
#  Futility bounds (non-binding)   : -Inf, -Inf
#  Cumulative alpha spending       : 0.0002592, 0.0071601, 0.0250000
#  Stage levels                    : 0.0002592, 0.0070554, 0.0225331
#  Effect sizes                    : 1.96, 1.86, NA
#  Test statistics                 : 3.228, 2.769, NA
#  p-values                        : 0.005177, 0.010895, NA
#  Overall test statistics         : 3.228, 4.342, NA
#  Overall p-values                : 0.0051766, 0.0001757, NA
#  Futility bounds for power       : NA
#  Actions                         : continue, reject and stop, NA
#  Theta H0                        : 0
#  CRP                             : 0.3177, 0.9434, NA
#  Planned sample size             : NA, NA, NA
#  Planned allocation ratio        : 1
#  Assumed effect                  : NA
#  Assumed standard deviation      : 1
#  Conditional power               : NA, NA, NA
#  RCIs (lower)                    : -1.236, 0.702, NA
#  RCIs (upper)                    : 5.16, 3.02, NA
#  Repeated p-values               : 0.081766, 0.001825, NA
#  Final stage                     : 2
#  Final p-value                   : NA, 0.0004094, NA
#  Final CIs (lower)               : NA, 0.654, NA
#  Final CIs (upper)               : NA, 2.36, NA
#  Median unbiased estimate        : NA, 1.51, NA
```

`getConditionalPower` *Get Conditional Power*

Description

Calculates and returns the conditional power.

Usage

```
getConditionalPower(design, stageResults, ..., nPlanned)
```

Arguments

`design`	The trial design.
`stageResults`	The results at given stage, obtained from `getStageResults`.
`nPlanned`	The sample size planned for the subsequent stages. It should be a vector with length equal to the remaining stages and is the overall sample size in the two treatment groups if two groups are considered.
`stage`	The stage number (optional). Default: total number of existing stages in the data input.
`allocationRatioPlanned`	The allocation ratio for two treatment groups planned for the subsequent stages, the default value is 1.
`thetaH1`	or pi1, pi2 A range of assumed effect sizes or assumed rates pi1 to calculate the conditional power. Depending on the type of dataset, either thetaH1 (means and survival) or pi1, pi2 (rates) can be specified. Additionally, if testing means is specified, an assumed standard (`assumedStDev`) deviation can be specified, default is 1.
`iterations`	Iterations for simulating the power for Fisher's combination test. If the power for more than one remaining stages is to be determined for Fisher's combination test, it is estimated via simulation with specified `iterations`, the default value is 10000.
`seed`	Seed for simulating the power for Fisher's combination test. See above, default is a random seed.

Details

The conditional power is calculated only if effect size and sample size is specified.

For Fisher's combination test, the conditional power for more than one remaining stages is estimated via simulation.

See Also

`StageResults_plot` or `AnalysisResults_plot` for plotting the conditional power.

getConditionalRejectionProbabilities *Get Conditional Rejection Probabilities*

Description

Calculates the conditional rejection probabilities (CRP) for given test results.

Usage

```
getConditionalRejectionProbabilities(design, stageResults, ...)
```

Arguments

design	The trial design.
stageResults	The results at given stage, obtained from getStageResults.
stage	The stage number (optional). Default: total number of existing stages in the data input.

Details

The conditional rejection probability is the probability, under H0, to reject H0 in one of the subsequent (remaining) stages. The probability is calculated using the specified design. For testing rates and the survival design, the normal approximation is used, i.e., it is calculated with the use of the prototype case testing a mean for normally distributed data with known variance.

The conditional rejection probabilities are provided up to the specified stage.

For Fisher's combination test, you can check the validity of the CRP calculation via simulation.

Examples

```
x <- getDesignFisher(kMax = 3, informationRates = c(0.1,0.8,1))
y <- getDataset(n = c(40,40), events = c(20,22))
getConditionalRejectionProbabilities(x, getStageResults(x, y, thetaH0 = 0.4))
# provides
# [1] 0.0216417 0.1068607        NA
```

getDataset *Get Dataset*

Description

Creates a dataset object and returns it.

Usage

```
getDataset(..., floatingPointNumbersEnabled = FALSE)
```

Arguments

`...`	A `data.frame` or some data vectors defining the dataset.
`floatingPointNumbersEnabled`	If `TRUE`, sample sizes can be specified as floating-point numbers (in general this only make sense for simulation purposes); by default `floatingPointNumbersEnabled = FALSE`, i.e., samples sizes defined as floating-point numbers will be truncated.

Details

The different dataset types `DatasetMeans`, of `DatasetRates`, or `DatasetSurvival` can be created as follows:

- An element of `DatasetMeans` for one sample is created by `getDataset(sampleSizes =, means =, stDevs =)` where `sampleSizes`, `means`, `stDevs` are vectors with stagewise sample sizes, means and standard deviations of length given by the number of available stages.
- An element of `DatasetMeans` for two samples is created by `getDataset(sampleSizes1 =, sampleSizes2 =, means1 =, means2 =, stDevs1 =, stDevs2 =)` where `sampleSizes1`, `sampleSizes2`, `means1`, `means2`, `stDevs1`, `stDevs2` are vectors with stagewise sample sizes, means and standard deviations for the two treatment groups of length given by the number of available stages.
- An element of `DatasetRates` for one sample is created by `getDataset(sampleSizes =, events =)` where `sampleSizes`, `events` are vectors with stagewise sample sizes and events of length given by the number of available stages.
- An element of `DatasetRates` for two samples is created by `getDataset(sampleSizes1 =, sampleSizes2 =, events1 =, events2 =)` where `sampleSizes1`, `sampleSizes2`, `events1`, `events2` are vectors with stagewise sample sizes and events for the two treatment groups of length given by the number of available stages.
- An element of `DatasetSurvival` is created by `getDataset(events=, logRanks =, allocationRatios =)` where `events`, `logRanks`, and `allocation ratios` are the stagewise events, (one-sided) logrank statistics, and allocation ratios.

Prefix `overall[Capital case of first letter of variable name]...` for the variable names enables entering the overall results and calculates stagewise statistics.

Note that in survival design usually the overall events and logrank test statistics are provided in the output, so `getDataset(overallEvents=, overallLogRanks =, overallAllocationRatios =)` is the usual command for entering survival data. Note also that for `overallLogranks` also the z scores from a Cox regression can be used.

`n` can be used in place of `samplesizes`.

Value

Returns a `Dataset` object.

Examples

```
# Create a Dataset of Means (one group):

datasetOfMeans <- getDataset(
   n = c(22, 11, 22, 11),
   means = c(1, 1.1, 1, 1),
   stDevs = c(1, 2, 2, 1.3)
)
datasetOfMeans
datasetOfMeans$show(showType = 2)

datasetOfMeans <- getDataset(
   overallSampleSizes = c(22, 33, 55, 66),
   overallMeans = c(1.000, 1.033, 1.020, 1.017 ),
   overallStDevs = c(1.00, 1.38, 1.64, 1.58)
)
datasetOfMeans
datasetOfMeans$show(showType = 2)
as.data.frame(datasetOfMeans)

# Create a Dataset of Means (two groups):

datasetOfMeans <- getDataset(
   n1 = c(22, 11, 22, 11),
   n2 = c(22, 13, 22, 13),
   means1 = c(1, 1.1, 1, 1),
   means2 = c(1.4, 1.5, 3, 2.5),
   stDevs1 = c(1, 2, 2, 1.3),
   stDevs2 = c(1, 2, 2, 1.3)
)
datasetOfMeans

datasetOfMeans <- getDataset(
   overallSampleSizes1 = c(22, 33, 55, 66),
   overallSampleSizes2 = c(22, 35, 57, 70),
   overallMeans1 = c(1, 1.033, 1.020, 1.017),
   overallMeans2 = c(1.4, 1.437, 2.040, 2.126),
   overallStDevs1 = c(1, 1.38, 1.64, 1.58),
   overallStDevs2 = c(1, 1.43, 1.82, 1.74)
)
datasetOfMeans

df <- data.frame(
   stages = 1:4,
   n1 = c(22, 11, 22, 11),
   n2 = c(22, 13, 22, 13),
   means1 = c(1, 1.1, 1, 1),
   means2 = c(1.4, 1.5, 3, 2.5),
   stDevs1 = c(1, 2, 2, 1.3),
   stDevs2 = c(1, 2, 2, 1.3)
)
datasetOfMeans <- getDataset(df)
datasetOfMeans

## Create a Dataset of Rates (one group):
```

```
datasetOfRates <- getDataset(
   n = c(8, 10, 9, 11),
   events = c(4, 5, 5, 6)
)
datasetOfRates

## Create a Dataset of Rates (two groups):

datasetOfRates <- getDataset(
   n2 = c(8, 10, 9, 11),
   n1 = c(11, 13, 12, 13),
   events2 = c(3, 5, 5, 6),
   events1 = c(10, 10, 12, 12)
)
datasetOfRates


## Create a Survival Dataset

dataset <- getDataset(
    overallEvents = c(8, 15, 19, 31),
   overallAllocationRatios = c(1, 1, 1, 2),
   overallLogRanks = c(1.52, 1.98, 1.99, 2.11)
)
dataset
```

`getDesignCharacteristics` *Get Design Characteristics*

Description

Calculates the characteristics of a design and returns it.

Usage

```
getDesignCharacteristics(design)
```

Arguments

`design`	The design.

Details

Calculates the inflation factor (IF), the expected reduction in sample size under H1, under H0, and under a value in between H0 and H1. Furthermore, absolute information values are calculated under the prototype case testing H0: mu = 0 against H1: mu = 1.

Value

Returns a `TrialDesignCharacteristics` object.

Examples

```
# Run with default values
getDesignCharacteristics(getDesignGroupSequential())
```

`getDesignFisher` *Get Design Fisher*

Description

Performs Fisher's combination test and returns critical values for this design.

Usage

```
getDesignFisher(..., kMax = NA_integer_, alpha = NA_real_,
  method = C_FISHER_METHOD_DEFAULT, userAlphaSpending = NA_real_,
  alpha0Vec = NA_real_, informationRates = NA_real_, sided = 1,
  bindingFutility = C_BINDING_FUTILITY_FISHER_DEFAULT,
  tolerance = C_ANALYSIS_TOLERANCE_FISHER_DEFAULT, iterations = 0,
  seed = NA_real_)
```

Arguments

`...`	Ensures that all arguments are be named and that a warning will be displayed if unknown arguments are passed.
`kMax`	The maximum number of stages K. K = 2, 3, ..., 6, default is 3.
`alpha`	The significance level alpha, default is 0.025.
`method`	"equalAlpha", "fullAlpha", "noInteraction", or "userDefinedAlpha", default is "equalAlpha".
`userAlphaSpending`	A vector of levels 0 < alpha_1 < ... < alpha_K < alpha specifying the cumulative Type I error rate.
`alpha0Vec`	Stopping for futility bounds for stage-wise p-values.
`informationRates`	Information rates that must be fixed prior to the trial, default is `(1 : kMax) / kMax`.
`sided`	Is the alternative one-sided (1) or two-sided (2), default is 1.
`bindingFutility`	If `bindingFutility = FALSE` is specified the calculation of the critical values is not affected by the futility bounds (default is `TRUE`).
`tolerance`	The tolerance, default is 1E-14.
`iterations`	The number of simulation iterations, e.g., getDesignFisher(iterations = 100000) checks the validity of the critical values for the default design. The default value of `iterations` is 0, i.e., no simulation will be executed.
`seed`	Seed for simulating the power for Fisher's combination test. See above, default is a random seed.

Details

`getDesignFisher` calculates the critical values and stage levels for Fisher's combination test as described in Bauer (1989), Bauer and Koehne (1994), Bauer and Roehmel (1995), and Wassmer (1999) for equally and unequally sized stages.

Value

Returns a `TrialDesignFisher` object

See Also

`getDesignSet` for creating a set of designs to compare.

Examples

```
# Run with default values
getDesignFisher()

# The output is:
#
# Design parameters and output of Fisher design:
# User defined parameters: not available
#
# Derived from user defined parameters: not available
#
# Default parameters:
#   Method                          : equalAlpha
#   Maximum number of stages        : 3
#   Stages                          : 1, 2, 3
#   Information rates               : 0.333, 0.667, 1.000
#   Significance level              : 0.0250
#   Alpha_0                         : 1.0000, 1.0000
#   Binding futility                : TRUE
#   Test                            : one-sided
#   Tolerance                       : 1e-14
#
# Output:
#   Cumulative alpha spending       : 0.01231, 0.01962, 0.02500
#   Critical values                 : 0.0123085, 0.0016636, 0.0002911
#   Stage levels                    : 0.01231, 0.01231, 0.01231
#   Scale                           : 1, 1
#   Non stochastic curtailment      : FALSE
```

`getDesignGroupSequential`

Get Design Group Sequential

Description

Provides adjusted boundaries and defines a group sequential design.

Usage

```
getDesignGroupSequential(..., kMax = NA_integer_, alpha = NA_real_,
  beta = NA_real_, sided = 1, informationRates = NA_real_,
  futilityBounds = NA_real_, typeOfDesign = C_DEFAULT_TYPE_OF_DESIGN,
  deltaWT = 0, optimizationCriterion = C_OPTIMIZATION_CRITERION_DEFAULT,
  gammaA = 1, typeBetaSpending = C_TYPE_OF_DESIGN_BS_NONE,
  userAlphaSpending = NA_real_, userBetaSpending = NA_real_, gammaB = 1,
  bindingFutility = C_BINDING_FUTILITY_DEFAULT,
  constantBoundsHP = C_CONST_BOUND_HP_DEFAULT,
  twoSidedPower = C_TWO_SIDED_POWER_DEFAULT,
  tolerance = C_DESIGN_TOLERANCE_DEFAULT)
```

Arguments

`...`	Ensures that all arguments are be named and that a warning will be displayed if unknown arguments are passed.
`kMax`	The maximum number of stages K. K = 1, 2, ..., 10, default is 3.
`alpha`	The significance level alpha, default is 0.025.
`beta`	Type II error rate, necessary for providing sample size calculations (e.g., `getSampleSizeMeans`), beta spending function designs, or optimum designs, default is 0.20.
`sided`	One-sided or two-sided, default is 1.
`informationRates`	The information rates, default is `(1 : kMax)/kMax`.
`futilityBounds`	The futility bounds (vector of length K - 1).
`typeOfDesign`	The type of design. Type of design is one of the following: O'Brien & Fleming ("OF"), Pocock ("P"), Wang & Tsiatis Delta class ("WT"), Haybittle & Peto ("HP"), Optimum design within Wang & Tsiatis class ("WToptimum"), O'Brien & Fleming type alpha spending ("OF"), Pocock type alpha spending ("asP"), Kim & DeMets alpha spending ("asKD"), Hwang, Shi & DeCani alpha spending ("asHSD"), user defined alpha spending ("asUser"), default is "OF".
`deltaWT`	Delta for Wang & Tsiatis Delta class.
`optimizationCriterion`	Optimization criterion for optimum design within Wang & Tsiatis class ("ASNH1", "ASNIFH1", "ASNsum"), default is "ASNH1".
`gammaA`	Parameter for alpha spending function, default is 1.
`typeBetaSpending`	Type of beta spending. Type of of beta spending is one of the following: O'Brien & Fleming type beta spending, Pocock type beta spending, Kim & DeMets beta spending, Hwang, Shi & DeCani beta spending, user defined beta spending ("bsOF", "bsP",...).
`userAlphaSpending`	The user defined alpha spending.
`userBetaSpending`	The user defined beta spending.
`gammaB`	Parameter for beta spending function, default is 1.

`bindingFutility`
: If `bindingFutility = TRUE` is specified the calculation of the critical values is affected by the futility bounds (default is FALSE).

`constantBoundsHP`
: The constant bounds up to stage K - 1 for the Haybittle & Peto design (default is 3).

`twoSidedPower`
: For two-sided testing, if `twoSidedPower = TRUE` is specified the sample size calculation is performed by considering both tails of the distribution. Default is FALSE, i.e., it is assumed that one tail probability is equal to 0 or the power should be directed to one part.

`tolerance`
: The tolerance, default is 1e-08.

Details

Depending on `typeOfDesign` some parameters are specified, others not. For example, only if `typeOfDesign` "asHSD" is selected, `gammaA` needs to be specified.

If an alpha spending approach was specified ("asOF", "asP", "asKD", "asHSD", or "asUser") additionally a beta spending function can be specified to produce futility bounds.

Value

Returns a `TrialDesignGroupSequential` object.

See Also

`getDesignSet` for creating a set of designs to compare.

Examples

```
# Run with default values
getDesignGroupSequential()

# The output is:
#
# Design parameters and output of group sequential design:
#
# User defined parameters: not available
#
# Derived from user defined parameters: not available
#
# Default parameters:
#   Type of design                        : OF
#   Maximum number of stages              : 3
#   Stages                                : 1, 2, 3
#   Information rates                     : 0.333, 0.667, 1.000
#   Significance level                    : 0.0250
#   Type II error rate                    : 0.2
#   Two-sided power                       : FALSE
#   Delta for Wang & Tsiatis Delta class  : 0
#   Futility bounds (non-binding)         : -Inf, -Inf
#   Binding futility                      : FALSE
#   Haybittle Peto constants              : 3.000
#   Parameter for alpha spending function : 1
```

```
#   Parameter for beta spending function  : 1
#   Optimization criterion for optimum design within Wang & Tsiatis class : ASNH1
#   Test                                  : one-sided
#   Tolerance                             : 1e-08
#   Type of beta spending                 : none
#
# Output:
#   Cumulative alpha spending             : 0.0002592, 0.0071601, 0.0250000
#   Critical values                       : 3.471, 2.454, 2.004
#   Stage levels                          : 0.0002592, 0.0070554, 0.0225331
#

# Calculate the Pocock type alpha spending critical values if the second
# interim analysis was performed after 70% of information was observed
getDesignGroupSequential(informationRates = c(0.4, 0.7),
    typeOfDesign = "asP")

# The output is:
#
# Design parameters and output of group sequential design :
# User defined parameters:
#   Type of design                        : asP
#   Stages                                : 1, 2
#   Information rates                     : 0.400, 0.700
#
# Derived from user defined parameters :
#   Maximum number of stages              : 2
#   Futility bounds (non-binding)         : -Inf
#
# Default parameters:
#   Significance level                    : 0.0250
#   Type II error rate                    : 0.2
#   Delta for Wang & Tsiatis Delta class  : 0
#   Parameter for alpha spending function : 1
#   Parameter for beta spending function  : 1
#   Optimization criterion for Optimum design within Wang & Tsiatis class : ASNH1
#   Test                                  : one-sided
#   Tolerance                             : 1e-08
#   Type of beta                          : none
# Output:
#   Cumulative alpha spending             : 0.01308, 0.01974
#   Critical values                       : 2.224, 2.305
#   Stage levels                          : 0.01308, 0.01058
```

`getDesignInverseNormal`

Get Design Inverse Normal

Description

Provides adjusted boundaries and defines a group sequential design for its use in the inverse normal combination test.

Usage

```
getDesignInverseNormal(..., kMax = NA_integer_, alpha = NA_real_,
  beta = NA_real_, sided = 1, informationRates = NA_real_,
  futilityBounds = NA_real_, typeOfDesign = C_DEFAULT_TYPE_OF_DESIGN,
  deltaWT = 0, optimizationCriterion = C_OPTIMIZATION_CRITERION_DEFAULT,
  gammaA = 1, typeBetaSpending = C_TYPE_OF_DESIGN_BS_NONE,
  userAlphaSpending = NA_real_, userBetaSpending = NA_real_, gammaB = 1,
  bindingFutility = C_BINDING_FUTILITY_DEFAULT,
  constantBoundsHP = C_CONST_BOUND_HP_DEFAULT,
  twoSidedPower = C_TWO_SIDED_POWER_DEFAULT,
  tolerance = C_DESIGN_TOLERANCE_DEFAULT)
```

Arguments

`...`	Ensures that all arguments are be named and that a warning will be displayed if unknown arguments are passed.
`kMax`	The maximum number of stages K. K = 1, 2, ..., 10, default is 3.
`alpha`	The significance level alpha, default is 0.025.
`beta`	Type II error rate, necessary for providing sample size calculations (e.g., `getSampleSizeMeans`), beta spending function designs, or optimum designs, default is 0.20.
`sided`	One-sided or two-sided, default is 1.
`informationRates`	The information rates, default is `(1 : kMax)/kMax`.
`futilityBounds`	The futility bounds (vector of length K - 1).
`typeOfDesign`	The type of design. Type of design is one of the following: O'Brien & Fleming ("OF"), Pocock ("P"), Wang & Tsiatis Delta class ("WT"), Haybittle & Peto ("HP"), Optimum design within Wang & Tsiatis class ("WToptimum"), O'Brien & Fleming type alpha spending ("OF"), Pocock type alpha spending ("asP"), Kim & DeMets alpha spending ("asKD"), Hwang, Shi & DeCani alpha spending ("asHSD"), user defined alpha spending ("asUser"), default is "OF".
`deltaWT`	Delta for Wang & Tsiatis Delta class.
`optimizationCriterion`	Optimization criterion for optimum design within Wang & Tsiatis class ("ASNH1", "ASNIFH1", "ASNsum"), default is "ASNH1".
`gammaA`	Parameter for alpha spending function, default is 1.
`typeBetaSpending`	Type of beta spending. Type of of beta spending is one of the following: O'Brien & Fleming type beta spending, Pocock type beta spending, Kim & DeMets beta spending, Hwang, Shi & DeCani beta spending, user defined beta spending ("bsOF", "bsP",...).
`userAlphaSpending`	The user defined alpha spending.
`userBetaSpending`	The user defined beta spending.
`gammaB`	Parameter for beta spending function, default is 1.

`bindingFutility`
: If `bindingFutility = TRUE` is specified the calculation of the critical values is affected by the futility bounds (default is FALSE).

`constantBoundsHP`
: The constant bounds up to stage K - 1 for the Haybittle & Peto design (default is 3).

`twoSidedPower`
: For two-sided testing, if `twoSidedPower = TRUE` is specified the sample size calculation is performed by considering both tails of the distribution. Default is FALSE, i.e., it is assumed that one tail probability is equal to 0 or the power should be directed to one part.

`tolerance`
: The tolerance, default is 1e-08.

Details

Depending on `typeOfDesign` some parameters are specified, others not. For example, only if `typeOfDesign` "asHSD" is selected, `gammaA` needs to be specified.

If an alpha spending approach was specified ("asOF", "asP", "asKD", "asHSD", or "asUser") additionally a beta spending function can be specified to produce futility bounds.

Value

Returns a `TrialDesignInverseNormal` object.

See Also

`getDesignSet` for creating a set of designs to compare.

Examples

```
# Run with default values
getDesignInverseNormal()

# The output is
#
# Design parameters and output of inverse normal design:
#
# User defined parameters: not available
#
# Derived from user defined parameters: not available
#
# Default parameters:
#   Type of design                        : OF
#   Maximum number of stages              : 3
#   Stages                                : 1, 2, 3
#   Information rates                     : 0.333, 0.667, 1.000
#   Significance level                    : 0.0250
#   Type II error rate                    : 0.2
#   Two-sided power                       : FALSE
#   Delta for Wang & Tsiatis Delta class  : 0
#   Futility bounds (non-binding)         : -Inf, -Inf
#   Binding futility                      : FALSE
#   Haybittle Peto constants              : 3.000
#   Parameter for alpha spending function : 1
```

```
#   Parameter for beta spending function  : 1
#   Optimization criterion for optimum design within Wang & Tsiatis class : ASNH1
#   Test                                  : one-sided
#   Tolerance                             : 1e-08
#   Type of beta spending                 : none
#
# Output:
#   Cumulative alpha spending             : 0.0002592, 0.0071601, 0.0250000
#   Critical values                       : 3.471, 2.454, 2.004
#   Stage levels                          : 0.0002592, 0.0070554, 0.0225331
#

# Calculate the Pocock type alpha spending critical values if the second
# interim analysis was performed after 70% of information was observed
getDesignInverseNormal(informationRates = c(0.4, 0.7),
    typeOfDesign = "asP")

# The output is
#
# Design parameters and output of inverse normal design:
#
# User defined parameters:
#   Type of design                        : asP
#   Stages                                : 1, 2
#   Information rates                     : 0.400, 0.700
#
# Derived from user defined parameters :
#   Maximum number of stages              : 2
#   Futility bounds (non-binding)         : -Inf
#
# Default parameters:
#   Significance level                    : 0.0250
#   Type II error rate                    : 0.2
#   Delta for Wang & Tsiatis Delta class  : 0
#   Parameter for alpha spending function : 1
#   Parameter for beta spending function  : 1
#   Optimization criterion for Optimum design within Wang & Tsiatis class : ASNH1
#   Test                                  : one-sided
#   Tolerance                             : 1e-08
#   Type of beta                          : none
# Output:
#   Cumulative alpha spending             : 0.01308, 0.01974
#   Critical values                       : 2.224, 2.305
#   Stage levels                          : 0.01308, 0.01058
```

`getDesignSet` *Get Design Set*

Description

Creates a trial design set object and returns it.

Usage

```
getDesignSet(...)
```

Arguments

...	'designs' OR 'design' and one or more design parameters, e.g., deltaWT = c(0.1, 0.3, 0.4). • `design` The master design (optional, you need to specify an additional parameter that shall be varied). • `designs` The designs to compare (optional).

Details

Specify a master design and one or more design parameters or a list of designs.

Value

Returns a `TrialDesignSet` object.

Examples

```
# Example 1
 design <- getDesignGroupSequential(alpha = 0.05, kMax = 6,
    sided = 2, typeOfDesign = "WT", deltaWT = 0.1)
 designSet <- getDesignSet()
 designSet$add(design = design, deltaWT = c(0.3, 0.4))
 if (require(ggplot2)) plot(designSet, type = 1)

# Example 2 (shorter script)
design <- getDesignGroupSequential(alpha = 0.05, kMax = 6,
    sided = 2, typeOfDesign = "WT", deltaWT = 0.1)
designSet <- getDesignSet(design = design, deltaWT = c(0.3, 0.4))
if (require(ggplot2)) plot(designSet)
```

`getFinalConfidenceInterval`

Get Final Confidence Interval

Description

Returns the final confidence interval for the parameter of interest. It is based on the prototype case, i.e., the test for testing a mean for normally distributed variables.

Usage

```
getFinalConfidenceInterval(design, dataInput, ...)
```

Arguments

`design`	The trial design.
`dataInput`	The data input.
`stage`	The stage number.

`thetaH0` The null hypothesis value, default is 0 for the normal and the binary case, it is 1 for the survival case. For testing a rate in one sample, a value `thetaH0` in (0,1) has to be specified for defining the null hypothesis H0: pi= thetaH0.
For noninferiority designs, this is the noninferiority bound.

`directionUpper`
The direction of one-sided testing. Default is `directionUpper = TRUE` which means that larger values of the test statistics yield smaller p-values.

`normalApproximation`
The type of computation of the p-values. Default is FALSE for testing means (i.e., the t test is used) and TRUE for testing rates and the hazard ratio. For testing rates, if `normalApproximation = FALSE` is specified, the binomial test (one sample) or the test of Fisher (two samples) is used for calculating the p-values. In the survival setting `normalApproximation = FALSE` has no effect.

`equalVariances`
The type of t test. For testing means in two treatment groups, either the t test assuming that the variances are equal or the t test without assuming this, i.e., the test of Welch-Satterthwaite is calculated, default is `equalVariances = TRUE`.

Details

Depending on `design` and `dataInput` the final confidence interval and median unbiased estimate that is based on the stagewise ordering of the sample space will be calculated and returned. Additionally, a non-standardized ("general") version is provided, use the standard deviation to obtain the confidence interval for the parameter of interest.

Value

Returns a `list` containing

- `finalStage`,
- `medianUnbiased`,
- `finalConfidenceInterval`,
- `medianUnbiasedGeneral`, and
- `finalConfidenceIntervalGeneral`.

Examples

```
design <- getDesignInverseNormal(kMax = 2)
data <- getDataset(
    n = c(20, 30),
    means = c(50, 51),
    stDevs = c(130, 140)
)
getFinalConfidenceInterval(design, dataInput = data)

# Results in:
#
# $finalStage
# [1] 2
#
# $medianUnbiasedGeneral
```

```
# [1] 0.3546145
#
# $finalConfidenceIntervalGeneral
# [1] 0.06967801 0.63468553
#
# $medianUnbiased
# [1] 47.7787
#
# $finalConfidenceInterval
# [1]  9.388012 85.513851'
```

`getFinalPValue` *Get Final P Value*

Description

Returns the final p-value for given stage results.

Usage

```
getFinalPValue(design, stageResults, ...)
```

Arguments

`design`	The trial design.
`stageResults`	The results at given stage, obtained from `getStageResults`.
`stage`	The stage number (optional). Default: total number of existing stages in the data input.

Details

The calculation of the final p-value is based on the stagewise ordering of the sample space. This enables the calculation for both the non-adaptive and the adaptive case. For Fisher's combination test, it is available for `kMax = 2` only.

`getPowerAndAverageSampleNumber`
Get Power And Average Sample Number

Description

Returns the power and average sample number of the specified design.

Usage

```
getPowerAndAverageSampleNumber(design, theta = seq(-1, 1, 0.02), nMax = 100)
```

Arguments

design	The design.
theta	A vector of theta values.
nMax	The maximum sample size.

Details

ASN = Average sample number (size)

---

getRepeatedConfidenceIntervals
Get Repeated Confidence Intervals

Description

Calculates and returns the lower and upper limit of the repeated confidence intervals of the trial.

Usage

```
getRepeatedConfidenceIntervals(design, dataInput, ...)
```

Arguments

design	The trial design.
dataInput	The summary data used for calculating the test results. This is either an element of DatasetMeans, of DatasetRates, or of DatasetSurvival. See getDataset.
stage	The stage number (optional). Default: total number of existing stages in the data input.

Details

The repeated confidence interval at a given stage of the trial contains the parameter values that are not rejected using the specified sequential design. It can be calculated at each stage of the trial and can thus be used as a monitoring tool.

The repeated confidence intervals are provided up to the specified stage.

getRepeatedPValues *Get Repeated P Values*

Description

Calculates the repeated p-values for given test results.

Usage

```
getRepeatedPValues(design, stageResults, ...)
```

Arguments

design	The trial design.
stageResults	The results at given stage, obtained from getStageResults.
stage	The stage number (optional). Default: total number of existing stages in the data input.

Details

The repeated p-value at a given stage of the trial is defined as the smallest significance level under which at given test design the test results obtain rejection of the null hypothesis. It can be calculated at each stage of the trial and can thus be used as a monitoring tool.

The repeated p-values are provided up to the specified stage.

getSampleSizeMeans *Get Sample Size Means*

Description

Returns the sample size for testing means in one and two samples.

Usage

```
getSampleSizeMeans(design, ..., normalApproximation = FALSE,
  meanRatio = FALSE, thetaH0 = 0, alternative = seq(0.2, 1, 0.2),
  stDev = 1, groups = 2, allocationRatioPlanned = 1)
```

Arguments

design	The trial design.
...	Ensures that all arguments are be named and that a warning will be displayed if unknown arguments are passed.
normalApproximation	If normalApproximation = TRUE is specified, the variance is assumed to be known, default is FALSE.
meanRatio	If meanRatio = TRUE is specified, the sample size for one-sided testing of H0: mu1/mu2 = thetaH0 is calculated, default is FALSE.

`thetaH0` The null hypothesis value. For one-sided testing, a value != 0 (or a value != 1 for testing the mean ratio) can be specified, default is 0.

`alternative` The alternative hypothesis value. This can be a vector of assumed alternatives, default is seq(0.1, 2, 0.2).

`stDev` The standard deviation, default is 1. If `meanRatio = TRUE` is specified, stDev defines the coefficient of variation sigma/mu2.

`groups` The number of treatment groups (1 or 2), default is 2.

`allocationRatioPlanned` The planned allocation ratio for a two treatment groups design, default is 1. If `allocationRatioPlanned = 0` is entered, the optimal allocation ratio yielding the smallest overall sample size is determined.

Details

At given design the function calculates the sample size for testing means. In a two treatment groups design, additionally, an allocation ratio = n1Fixed/n2Fixed can be specified. A null hypothesis value thetaH0 != 0 for testing the difference of two means or thetaH0 != 1 for testing the ratio of two means can be specified.

Value

Returns a `TrialDesignPlanMeans` object.

Examples

```
# Calculate sample sizes n1Fixed, n2Fixed, and
# nFixed for a range of alternative values:
getSampleSizeMeans(getDesignGroupSequential(alpha = 0.025, sided = 1),
    groups = 2, alternative = seq(0.1, 2, 0.2),
    normalApproximation = FALSE, allocationRatioPlanned = 2)
```

`getSampleSizeRates` *Get Sample Size Rates*

Description

Returns the sample size for testing rates in one and two samples.

Usage

```
getSampleSizeRates(design, ..., normalApproximation = TRUE,
  riskRatio = FALSE, thetaH0 = 0, pi1 = seq(0.4, 0.6, 0.1), pi2 = 0.2,
  groups = 2, allocationRatioPlanned = 1)
```

Arguments

`design`	The trial design.
`...`	Ensures that all arguments are be named and that a warning will be displayed if unknown arguments are passed.
`normalApproximation`	If `normalApproximation = FALSE` is specified, the sample size for the case of one treatment group is calculated exactly using the binomial distribution. default is TRUE.
`riskRatio`	If `riskRatio = TRUE` is specified, the sample size for one-sided testing of H0: pi1/pi2 = thetaH0 is calculated, default is FALSE.
`thetaH0`	The null hypothesis value. For one-sided testing, a value != 0 (or != 1 for testing the risk ratio pi1/pi2) can be specified, default is 0.
`pi1`	The assumed probability in the treatment group if two treatment groups are considered, or the alternative probability for a one treatment group design, default is seq(0.4, 0.6, 0.1).
`pi2`	The assumed probability in the control group if two treatment groups are considered, default is 0.2.
`groups`	The number of treatment groups (1 or 2), default is 2.
`allocationRatioPlanned`	The planned allocation ratio for a two treatment groups design. If `allocationRatioPlanned = 0` is entered, the optimal allocation ratio yielding the smallest overall sample size is determined, default is 1.

Details

At given design the function calculates the sample size for testing rates. In a two treatment groups design, additionally, an allocation ratio = n1Fixed/n2Fixed can be specified. If a null hypothesis value thetaH0 != 0 for testing the difference of two rates thetaH0 != 1 for testing the risk ratio is specified, the sample size formula according to Farrington & Manning (Statistics in Medicine, 1990) is used.

Value

Returns a `TrialDesignPlanRates` object.

Examples

```
# Calculate sample sizes n1Fixed, n2Fixed, nFixed, and the optimum
# allocation ratios for a range of pi1 values when testing
# H0: pi1 - pi2 = -0.1:
getSampleSizeRates(getDesignGroupSequential(alpha = 0.025, beta = 0.2,
    sided = 1), groups = 2, thetaH0 = -0.1, pi1 = seq(0.4, 0.55, 0.025),
    pi2 = 0.4, normalApproximation = TRUE, allocationRatioPlanned = 0)

# Calculate sample sizes n1Fixed, n2Fixed, nFixed, and the optimum
# allocation ratios for a range of pi2 values when testing
# H0: pi1 / pi2 = 1.25:
getSampleSizeRates(getDesignGroupSequential(alpha = 0.025, beta = 0.2,
    sided = 1), groups = 2, riskRatio = TRUE, thetaH0 = 1.25, pi1 = 0.3,
    pi2 = 0.3, normalApproximation = TRUE, allocationRatioPlanned = 1)
```

`getSampleSizeSurvival` *Get Sample Size Survival*

Description

Returns the sample size for testing the hazard ratio in a two treatment groups survival design.

Usage

```
getSampleSizeSurvival(design, ..., typeOfComputation = "Schoenfeld",
  thetaH0 = 1, pi2 = 0.2, pi1 = seq(0.4, 0.6, 0.1),
  allocationRatioPlanned = 1, accountForObservationTimes = NA,
  eventTime = NA_real_, accrualTime = NA_real_, followUpTime = NA_real_,
  maxNumberOfPatients = 0, dropOutRate1 = 0, dropOutRate2 = 0,
  dropOutTime = NA_real_)
```

Arguments

`design`	The trial design.
`...`	Ensures that all arguments are be named and that a warning will be displayed if unknown arguments are passed.
`typeOfComputation`	Three options are available: "Schoenfeld", "Freedman", "HsiehFreedman", the default is "Schoenfeld". For details, see Hsieh (Statistics in Medicine, 1992).
`thetaH0`	The null hypothesis value. The default value is 1. For one-sided testing, a bound for testing H0: log(1 - pi1)/log(1 - pi2) = thetaH0 != 1 can be specified.
`pi2`	The assumed event rate in the control group, default is 0.2.
`pi1`	The assumed event rate in the treatment group, default is seq(0.4, 0.6, 0.1).
`allocationRatioPlanned`	The planned allocation ratio, default is 1. If `allocationRatioPlanned = 0` is entered, the optimal allocation ratio yielding the smallest number of patients is determined.
`accountForObservationTimes`	If `accountForObservationTimes = TRUE`, the number of patients is calculates assuming assuming specific accrual and follow-up time. The formula of Kim & Tsiatis (Biometrics, 1990) is used to calculated the expected number of events under the alternative (see also Lakatos & Lan, Statistics in Medicine, 1992).
`eventTime`	The assumed time under which the event rates are calculated (need to be specified if `accountForObservationTimes = TRUE`).
`accrualTime`	The assumed accrual-up time for the study (need to be specified if `accountForObservationTimes = TRUE`).
`followUpTime`	The assumed (additional) follow-up time for the study (need to be specified if `accountForObservationTimes = TRUE`). The total study duration is accrualTime + followUpTime.

`maxNumberOfPatients`
: For `accountForObservationTimes = TRUE`, if `maxNumberOfPatients > 0` is specified, the follow-up time for the required number of events is determined.

`dropOutRate1` The assumed drop-out rate in the control group, default is 0.

`dropOutRate2` The assumed drop-out rate in the treatment group, default is 0.

`dropOutTime` The assumed time for drop-out rates in the control and the treatment group, default is 12.

Details

At given design the function calculates the number of events and an estimate for the necessary number of patients for testing the hazard ratio log(1 - p1)/log(1 - p2) with no interim stages. Accordingly, an event probability omega is calculated. It also calculates the time when the required events are expected under the given assumptions (assuming exponentially distributed survival times). Furthermore, an allocation ratio = n1Fixed/n2Fixed can be specified.

Value

Returns a `TrialDesignPlanSurvival` object.

Examples

```
# Calculate the number of events and number of patients calculated with
# the Schoenfeld formula.
getSampleSizeSurvival(getDesignGroupSequential(alpha = 0.025, beta = 0.2,
    sided = 1), thetaH0 = 1, pi1 = 0.6, pi2 = 0.9,
    allocationRatioPlanned = 2, typeOfComputation = "Schoenfeld")

# Calculate analysis times, number of aevent, and number of patients
# under specified event, accrual, followup, and dropout time and event
# and dropout rates.
getSampleSizeSurvival(getDesignGroupSequential(alpha = .025, sided = 1),
    pi1 = c(0.25, 0.3, 0.35), pi2 = 0.4, allocationRatioPlanned = 0,
    typeOfComputation = "Schoenfeld", accountForObservationTimes = TRUE,
    eventTime = 12, accrualTime = 6, followUpTime = 12,
    maxNumberOfPatients = 0, dropOutRate1 = 0.15, dropOutRate2 = 0.1,
    dropOutTime = 24)
```

---

getStageResults *Get Stage Results*

Description

Returns summary statistics and p-values for a given data set and a given design.

Usage

```
getStageResults(design, dataInput, ...)
```

Arguments

`design` The trial design.

`dataInput` The summary data used for calculating the test results. This is either an element of `DatasetMeans`, of `DatasetRates`, or of `DatasetSurvival`. See `getDataset`.

`...` Further (optional) arguments to be passed:

stage The stage number (optional). Default: total number of existing stages in the data input.

thetaH0 The null hypothesis value, default is 0 for the normal and the binary case, it is 1 for the survival case. For testing a rate in one sample, a value thetaH0 in (0, 1) has to be specified for defining the null hypothesis H0: pi = thetaH0.
For noninferiority designs, this is the noninferiority bound.

thetaH1 and assumedStDev or pi1, pi2 The assumed effect size or assumed rates to calculate the conditional power. Depending on the type of dataset, either thetaH1 (means and survival) or pi1, pi2 (rates) can be specified. Additionally, if testing means is specified, an assumed standard deviation can be specified, default is 1.

normalApproximation The type of computation of the p-values. Default is FALSE for testing means (i.e., the t test is used) and TRUE for testing rates and the hazard ratio. For testing rates, if `normalApproximation = FALSE` is specified, the binomial test (one sample) or the test of Fisher (two samples) is used for calculating the p-values. In the survival setting, `normalApproximation = FALSE` has no effect.

equalVariances The type of t test. For testing means in two treatment groups, either the t test assuming that the variances are equal or the t test without assuming this, i.e., the test of Welch-Satterthwaite is calculated, default is `equalVariances = TRUE`.

directionUpper The direction of one-sided testing. Default is `directionUpper = TRUE` which means that larger values of the test statistics yield smaller p-values.

Details

Calculates and returns the stage results of the specified design and data input at the specified stage.

Value

Returns a `StageResults` object.

Examples

```
design <- getDesignInverseNormal()
dataRates <- getDataset(
    n1 = c(10,10),
    n2 = c(20,20),
    events1 = c(8,10),
    events2 = c(10,16))

getStageResults(design, dataRates)

# produces:
```

```
#
# Stage results of rates:
#   Stages                      : 1, 2, 3
#   Overall test statistics     : 1.581, 2.064, NA
#   Overall p-values            : 0.05692, 0.01949, NA
#   Overall events (1)          : 8, 18
#   Overall events (2)          : 10, 26
#   Overall sample sizes (1)    : 10, 20
#   Overall sample sizes (2)    : 20, 40
#   Test statistics             : 1.581, 1.519, NA
#   p-values                    : 0.05692, 0.06437, NA
#   Effect sizes                : 0.30, 0.25, NA
#   Inverse Normal Combination  : 1.581, 2.192, NA
#   Weights Inverse Normal      : 0.577, 0.577, 0.577
#   Theta H0                    : 0
#   Direction                   : upper
#   Normal approximation        : TRUE
```

---

getTestActions *Get Test Actions*

Description

Returns test actions.

Usage

```
getTestActions(design, stageResults, ...)
```

Arguments

design	The trial design.
stageResults	The results at given stage, obtained from getStageResults.
stage	The stage number (optional). Default: total number of existing stages in the data input.

Details

Returns the test actions of the specified design and stage results at the specified stage.

ParameterSet *Parameter Set*

Description

Basic class for parameter sets.

Details

The parameter set implements basic functions for a set of parameters.

ParameterSet_as.data.frame *Coerce Parameter Set to a Data Frame*

Description

Returns the `ParameterSet` as data frame.

Usage

```
## S3 method for class 'ParameterSet'
as.data.frame(x, row.names = NULL, optional = FALSE,
  niceColumnNamesEnabled = TRUE, includeAllParameters = FALSE, ...)
```

Details

Coerces the parameter set to a data frame.

ParameterSet_print *Print Parameter Set Values*

Description

`print` prints its `ParameterSet` argument and returns it invisibly (via `invisible(x)`).

Usage

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

Details

Prints the parameters and results of a parameter set.

ParameterSet_summary *Parameter Set Summary*

Description

Displays a summary of `ParameterSet` object.

Usage

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

Details

Summarizes the parameters and results of a parameter set.

PlotSettings *Plot Settings*

Description

Class for plot settings.

Details

Collects typical plot settings in an object.

Fields

`lineSize` The line size.

`pointSize` The point size.

`mainTitleFontSize` The main tile font size.

`axesTextFontSize` The text font size.

`legendFontSize` The legend font size.

Methods

`adjustLegendFontSize(adjustingValue)` Adjusts the legend font size, e.g., run
`adjustLegendFontSize(-2)` # makes the font size 2 points smaller

`enlargeAxisTicks(p)` Enlarges the axis ticks

`expandAxesRange(p, x = NA_real_, y = NA_real_)` Expands the axes range

`hideGridLines(p)` Hides the grid lines

`setAxesAppearance(p)` Sets the font size and face of the axes titles and texts

`setColorPalette(p, palette, mode = c("colour", "fill", "all"))` Sets the color palette

`setLegendBorder(p)` Sets the legend border

`setMainTitle(p, mainTitle, subtitle = NA_character_)` Sets the main title

`setMarginAroundPlot(p, margin = 0.2)` Sets the margin around the plot, e.g., run
`setMarginAroundPlot(p, .2)` or
`setMarginAroundPlot(p, c(.1, .2, .1, .2)`

`setTheme(p)` Sets the theme

PowerAndAverageSampleNumberResult
Power and Average Sample Number Result

Description

Class for power and average sample number (ASN) results.

Details

This object can not be created directly; use `getPowerAndAverageSampleNumber` with suitable arguments to create it.

PowerAndAverageSampleNumberResult_as.data.frame *Coerce Power And Average Sample Number Result to a Data Frame*

Description

Returns the PowerAndAverageSampleNumberResult as data frame.

Usage

```
## S3 method for class 'PowerAndAverageSampleNumberResult'
as.data.frame(x, row.names = NULL,
  optional = FALSE, niceColumnNamesEnabled = TRUE,
  includeAllParameters = FALSE, ...)
```

Details

Coerces the object to a data frame.

readDataset *Read Dataset*

Description

Reads a data file and returns it as dataset object.

Usage

```
readDataset(file, ..., header = TRUE, sep = ",", quote = "\"",
  dec = ".", fill = TRUE, comment.char = "", fileEncoding = "UTF-8")
```

Arguments

file	A CSV file (see read.table).
...	Further arguments to be passed to coderead.table.
header	A logical value indicating whether the file contains the names of the variables as its first line.
sep	The field separator character. Values on each line of the file are separated by this character. If sep = "," (the default for readDataset) the separator is a comma.
quote	The set of quoting characters. To disable quoting altogether, use quote = "". See scan for the behaviour on quotes embedded in quotes. Quoting is only considered for columns read as character, which is all of them unless colClasses is specified.
dec	The character used in the file for decimal points.
fill	logical. If TRUE then in case the rows have unequal length, blank fields are implicitly added.

`comment.char` character: a character vector of length one containing a single character or an empty string. Use "" to turn off the interpretation of comments altogether.

`fileEncoding` character string: if non-empty declares the encoding used on a file (not a connection) so the character data can be re-encoded. See the 'Encoding' section of the help for file, the 'R Data Import/Export Manual' and 'Note'.

Details

`readDataset` is a wrapper function that uses `read.table` to read the CSV file into a data frame, transfers it from long to wide format with `reshape` and puts the data to `getDataset`.

Value

Returns a `Dataset` object.

See Also

- `readDatasets` for reading multiple datasets,
- `writeDataset` for writing a single dataset,
- `writeDatasets` for writing multiple datasets.

`readDatasets` *Read Multiple Datasets*

Description

Reads a data file and returns it as a list of dataset objects.

Usage

```
readDatasets(file, ..., header = TRUE, sep = ",", quote = "\"",
  dec = ".", fill = TRUE, comment.char = "", fileEncoding = "UTF-8")
```

Arguments

`file`	A CSV file (see `read.table`).
`...`	Further arguments to be passed to coderead.table.
`header`	A logical value indicating whether the file contains the names of the variables as its first line.
`sep`	The field separator character. Values on each line of the file are separated by this character. If sep = "," (the default for `readDatasets`) the separator is a comma.
`quote`	The set of quoting characters. To disable quoting altogether, use quote = "". See scan for the behaviour on quotes embedded in quotes. Quoting is only considered for columns read as character, which is all of them unless `colClasses` is specified.
`dec`	The character used in the file for decimal points.
`fill`	logical. If `TRUE` then in case the rows have unequal length, blank fields are implicitly added.

`comment.char` character: a character vector of length one containing a single character or an empty string. Use "" to turn off the interpretation of comments altogether.

`fileEncoding` character string: if non-empty declares the encoding used on a file (not a connection) so the character data can be re-encoded. See the 'Encoding' section of the help for file, the 'R Data Import/Export Manual' and 'Note'.

Details

Reads a file that was written by `writeDatasets` before.

Value

Returns a list of `Dataset` objects.

See Also

- `readDataset` for reading a single dataset,
- `writeDatasets` for writing multiple datasets,
- `writeDataset` for writing a single dataset.

`StageResults` *Basic Stage Results*

Description

Basic class for stage results.

Details

`StageResults` is the basic class for `StageResultsMeans`, `StageResultsRates`, and `StageResultsSurvival`.

Fields

`testStatistics` The stage-wise test statistics.

`pValues` The stage-wise p-values.

`combInverseNormal` The inverse normal test.

`combFisher` The Fisher's combination test.

`effectSizes` The effect sizes for different designs.

`testActions` The action drawn from test result.

`weightsFisher` The weights for Fisher's combination test.

`weightsInverseNormal` The weights for inverse normal statistic.

StageResultsMeans *Stage Results of Means*

Description

Class for stage results of means.

Details

This object can not be created directly; use `getStageResults` with suitable arguments to create the stage results of a dataset of means.

Fields

`testStatistics` The stage-wise test statistics.

`pValues` The stage-wise p-values.

`combInverseNormal` The inverse normal test.

`combFisher` The Fisher's combination test.

`effectSizes` The effect sizes for different designs.

`testActions` The action drawn from test result.

`weightsFisher` The weights for Fisher's combination test.

`weightsInverseNormal` The weights for inverse normal statistic.

StageResultsRates *Stage Results of Rates*

Description

Class for stage results of rates.

Details

This object can not be created directly; use `getStageResults` with suitable arguments to create the stage results of a dataset of rates.

Fields

`testStatistics` The stage-wise test statistics.

`pValues` The stage-wise p-values.

`combInverseNormal` The inverse normal test.

`combFisher` The Fisher's combination test.

`effectSizes` The effect sizes for different designs.

`testActions` The action drawn from test result.

`weightsFisher` The weights for Fisher's combination test.

`weightsInverseNormal` The weights for inverse normal statistic.

StageResultsSurvival
Stage Results of Survival Data

Description

Class for stage results survival data.

Details

This object can not be created directly; use `getStageResults` with suitable arguments to create the stage results of a dataset of survival data.

Fields

`testStatistics` The stage-wise test statistics.

`pValues` The stage-wise p-values.

`combInverseNormal` The inverse normal test.

`combFisher` The Fisher's combination test.

`effectSizes` The effect sizes for different designs.

`testActions` The action drawn from test result.

`weightsFisher` The weights for Fisher's combination test.

`weightsInverseNormal` The weights for inverse normal statistic.

StageResults_as.data.frame
Coerce Stage Results to a Data Frame

Description

Returns the `StageResults` as data frame.

Usage

```
## S3 method for class 'StageResults'
as.data.frame(x, row.names = NULL, optional = FALSE,
  niceColumnNamesEnabled = TRUE, includeAllParameters = FALSE, type = 1,
  ...)
```

Details

Coerces the stage results to a data frame.

StageResults_names *The Names of a Stage Results object*

Description

Function to get the names of a `StageResults` object.

Usage

```
## S3 method for class 'StageResults'
names(x)
```

Details

Returns the names of stage results that can be accessed by the user.

StageResults_plot *Stage Results Plotting - Conditional Power Plot*

Description

Plots the conditional power together with the likelihood function.

Usage

```
## S3 method for class 'StageResults'
plot(x, y, ..., nPlanned,
  stage = x$getNumberOfStages(), allocationRatioPlanned = NA_real_,
  main = NA_character_, xlab = NA_character_, ylab = NA_character_,
  legendTitle = NA_character_, palette = "Set1",
  legendPosition = NA_integer_, type = 1)
```

Arguments

`x`	The stage results at given stage, obtained from `getStageResults`.
`y`	Not available for this kind of plot (is only defined to be compatible to the generic plot function).
`...`	Optional `ggplot2` arguments. Furthermore the following arguments can be defined: • `directionUpper`: The direction of one-sided testing. Default is `directionUpper = TR` which means that larger values of the test statistics yield smaller p-values. • `thetaH0`: The null hypothesis value, default is 0 for the normal and the binary case, it is 1 for the survival case. For testing a rate in one sample, a value thetaH0 in (0,1) has to be specified for defining the null hypothesis H0: pi = thetaH0.

- `thetaH1` or `pi1`, `pi2`: A range of assumed effect sizes or assumed rates pi2 to calculate the conditional power. Depending on the type of dataset, either thetaH1 (means and survival) or pi1, pi2 (rates) can be specified. Additionally, if testing means is specified, an assumed standard deviation can be specified (default is 1).

`nPlanned` The sample size planned for the subsequent stages. It should be a vector with length equal to the remaining stages and is the overall sample size in the two treatment groups if two groups are considered.

`stage` The stage number (optional). Default: total number of existing stages in the data input used to create the stage results.

`allocationRatioPlanned`
The allocation ratio for two treatment groups planned for the subsequent stages, the default value is 1.

`main` The main title.

`xlab` The x-axis label.

`ylab` The y-axis label.

`legendTitle` The legend title.

`palette` The palette, default is `"Set1"`.

`legendPosition`
The position of the legend. By default (`NA_integer_`) the algorithm tries to find a suitable position. Choose one of the following values to specify the position manually:

- `0`: legend position outside plot
- `1`: legend position left top
- `2`: legend position left center
- `3`: legend position left bottom
- `4`: legend position right top
- `5`: legend position right center
- `6`: legend position right bottom

`type` The plot type (default = 1). Note that at the moment only one type (the conditional power plot) is available.

Details

Generic function to plot all kinds of stage results. The conditional power is calculated only if effect size and sample size is specified.

Examples

```
design <- getDesignGroupSequential(kMax = 4, alpha = 0.025,
    informationRates = c(0.2, 0.5, 0.8, 1),
    typeOfDesign = "WT", deltaWT = 0.25)

dataExample <- getDataset(
    n = c(20, 30, 30),
    means = c(0.5, 0.51, 0.55) * 100,
    stDevs = c(1.3, 1.4, 1.2) * 100
)
```

```
stageResults <- getStageResults(design, dataExample, thetaH0 = 20)

if (require(ggplot2)) plot(stageResults, nPlanned = c(30), thetaRange = c(0, 100))
```

StageResults_print *Print Stage Results*

Description

`print` prints its `StageResults` argument and returns it invisibly (via `invisible(x)`).

Usage

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

Details

Prints the parameters and values of stage results.

StageResults_summary *Stage Results Summary*

Description

Displays a summary of `StageResults` object.

Usage

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

Details

Summarizes the parameters and results of stage results.

TrialDesign *Basic Trial Design*

Description

Basic class for trial designs.

Details

`TrialDesign` is the basic class for

- `TrialDesignFisher`,
- `TrialDesignGroupSequential`, and
- `TrialDesignInverseNormal`.

TrialDesignCharacteristics
Trial Design Characteristics

Description

Class for trial design characteristics.

Details

`TrialDesignCharacteristics` contains all fields required to collect the characteristics of a design. This object should not be created directly; use `getDesignCharacteristics` with suitable arguments to create it.

See Also

`getDesignCharacteristics` for getting the design characteristics.

TrialDesignCharacteristics_as.data.frame
Coerce TrialDesignCharacteristics to a Data Frame

Description

Returns the `TrialDesignCharacteristics` as data frame.

Usage

```
## S3 method for class 'TrialDesignCharacteristics'
as.data.frame(x, row.names = NULL,
  optional = FALSE, niceColumnNamesEnabled = TRUE,
  includeAllParameters = FALSE, ...)
```

Arguments

`niceColumnNamesEnabled`
: logical. If `TRUE`, nice looking names will be used; syntactic names otherwise (see `make.names`).

`includeAllParameters`
: logical. If `TRUE`, all parameters will be included; a meaningful parameter selection otherwise.

Details

Each element of the `TrialDesignCharacteristics` is converted to a column in the data frame.

TrialDesignFisher *Fisher Design*

Description

Trial design for Fisher's combination test.

Details

This object should not be created directly; use `getDesignFisher` with suitable arguments to create a Fisher design.

See Also

`getDesignFisher` for creating a Fisher design.

TrialDesignGroupSequential *Group Sequential Design*

Description

Trial design for group sequential design.

Details

This object should not be created directly; use `getDesignGroupSequential` with suitable arguments to create a group sequential design.

Methods

`show(showType = 1)` Method for automatically printing trial design objects

See Also

`getDesignGroupSequential` for creating a group sequential design.

TrialDesignInverseNormal *Inverse Normal Design*

Description

Trial design for inverse normal method.

Details

This object should not be created directly; use `getDesignInverseNormal` with suitable arguments to create a inverse normal design.

See Also

`getDesignInverseNormal` for creating a inverse normal design.

TrialDesignPlan *Basic Trial Design Plan*

Description

Basic class for trial design plans.

Details

`TrialDesignPlan` is the basic class for

- `TrialDesignPlanMeans`,
- `TrialDesignPlanRates`, and
- `TrialDesignPlanSurvival`.

TrialDesignPlanMeans *Trial Design Plan Means*

Description

Trial design plan for means.

Details

This object can not be created directly; use `getSampleSizeMeans` with suitable arguments to create a design plan for a dataset of means.

Methods

`show(showType = 1)` Method for automatically printing trial plan objects

TrialDesignPlanRates
Trial Design Plan Rates

Description

Trial design plan for rates.

Details

This object can not be created directly; use `getSampleSizeRates` with suitable arguments to create a design plan for a dataset of rates.

Methods

`show(showType = 1)` Method for automatically printing trial plan objects

TrialDesignPlanSurvival
Trial Design Plan Survival

Description

Trial design plan for survival data.

Details

This object can not be created directly; use `getSampleSizeSurvival` with suitable arguments to create a design plan for a dataset of survival data.

Methods

`show(showType = 1)` Method for automatically printing trial plan objects

TrialDesignPlan_as.data.frame
Coerce Trial Design Plan to a Data Frame

Description

Returns the `TrialDesignPlan` as data frame.

Usage

```
## S3 method for class 'TrialDesignPlan'
as.data.frame(x, row.names = NULL,
  optional = FALSE, niceColumnNamesEnabled = TRUE,
  includeAllParameters = FALSE, ...)
```

Details

Coerces the design plan to a data frame.

`TrialDesignPlan_print` *Print Trial Design Plan*

Description

`print` prints its `TrialDesignPlan` argument and returns it invisibly (via `invisible(x)`).

Usage

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

Details

Prints the parameters and results of a design plan.

`TrialDesignSet` *Class for trial design sets.*

Description

`TrialDesignSet` is a class for creating a collection of different trial designs.

Details

This object can not be created directly; better use `getDesignSet` with suitable arguments to create a set of designs.

Fields

`designs` The designs (optional).

`design` The master design (optional).

Methods

`add(...)` Adds 'designs' OR a 'design' and/or a design parameter, e.g., deltaWT = c(0.1, 0.3, 0.4)

See Also

`getDesignSet`

`TrialDesignSet_as.data.frame`
Coerce Trial Design Set to a Data Frame

Description

Returns the `TrialDesignSet` as data frame.

Usage

```
## S3 method for class 'TrialDesignSet'
as.data.frame(x, row.names = NULL,
  optional = FALSE, niceColumnNamesEnabled = TRUE,
  includeAllParameters = FALSE, addPowerAndAverageSampleNumber = FALSE,
  theta = seq(-1, 1, 0.02), nMax = NA_integer_, ...)
```

Details

Coerces the design set to a data frame.

`TrialDesignSet_length`
Length of Trial Design Set

Description

Returns the number of designs in a `TrialDesignSet`.

Usage

```
## S3 method for class 'TrialDesignSet'
length(x)
```

Details

Is helpful for iteration over all designs in a design set with "[index]"-syntax.

TrialDesignSet_names
The Names of a Trial Design Set object

Description

Function to get the names of a `TrialDesignSet` object.

Usage

```
## S3 method for class 'TrialDesignSet'
names(x)
```

Details

Returns the names of a design set that can be accessed by the user.

TrialDesignSet_plot
Trial Design Set Plotting

Description

Plots a trial design set.

Usage

```
## S3 method for class 'TrialDesignSet'
plot(x, y, type = 1L, main = NA_character_,
  xlab = NA_character_, ylab = NA_character_, palette = "Set1",
  theta = seq(-1, 1, 0.02), nMax = NA_integer_, plotPointsEnabled = NA,
  legendPosition = NA_integer_, ...)
```

Arguments

`x`	The trial design set, obtained from `getDesignSet`.
`y`	Not available for this kind of plot (is only defined to be compatible to the generic plot function).
`type`	The plot type (default = `1`). The following plot types are available: • `1`: creates a 'Boundary Plot' • `2`: creates an 'Average Sample Size and Power / Early Stop' plot • `3`: creates a 'Stage Levels Plot' • `4`: creates a 'Power' plot • `5`: creates a 'Stopping Probability' plot • `6`: creates an 'Error Spending Plot'
`main`	The main title.
`xlab`	The x-axis label.

`ylab` The y-axis label.

`palette` The palette, default is `"Set1"`.

`theta` A vector of theta values.

`nMax` The maximum sample size.

`plotPointsEnabled`
If `TRUE`, additional points will be plotted.

`legendPosition`
The position of the legend. By default (`NA_integer_`) the algorithm tries to find a suitable position. Choose one of the following values to specify the position manually:

- `-1`: no legend will be shown
- `NA`: the algorithm tries to find a suitable position
- `0`: legend position outside plot
- `1`: legend position left top
- `2`: legend position left center
- `3`: legend position left bottom
- `4`: legend position right top
- `5`: legend position right center
- `6`: legend position right bottom

`...` Optional `ggplot2` arguments.

Details

Generic function to plot a trial design set. Is, e.g., useful to compare different designs or design parameters visual.

Value

Returns a `ggplot2` object.

Examples

```
design <- getDesignInverseNormal(kMax = 3, alpha = 0.025,
      typeOfDesign = "asKD", gammaA = 2,
      informationRates = c(0.2, 0.7, 1), typeBetaSpending = "bsOF")

 # Create a set of designs based on the master design defined above
 # and varied parameter 'gammaA'
  designSet <- getDesignSet(design = design, gammaA = c(3, 4))

  if (require(ggplot2)) plot(designSet, type = 1, legendPosition = 6)
```

TrialDesignSet_print
Print Trial Design Set Values

Description

`print` prints its `TrialDesignSet` argument and returns it invisibly (via `invisible(x)`).

Usage

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

Details

Prints the design set.

TrialDesign_as.data.frame
Coerce TrialDesign to a Data Frame

Description

Returns the `TrialDesign` as data frame.

Usage

```
## S3 method for class 'TrialDesign'
as.data.frame(x, row.names = NULL, optional = FALSE,
  niceColumnNamesEnabled = TRUE, includeAllParameters = FALSE, ...)
```

Arguments

`niceColumnNamesEnabled`
: logical. If `TRUE`, nice looking names will be used; syntactic names otherwise (see `make.names`).

`includeAllParameters`
: logical. If `TRUE`, all parameters will be included; a meaningful parameter selection otherwise.

Details

Each element of the `TrialDesign` is converted to a column in the data frame.

`TrialDesign_plot` *Trial Design Plotting*

Description

Plots a trial design.

Usage

```
## S3 method for class 'TrialDesign'
plot(x, y, main = NA_character_, xlab = NA_character_,
  ylab = NA_character_, type = 1, palette = "Set1", theta = seq(-2, 2,
  0.01), nMax = NA_integer_, plotPointsEnabled = NA,
  legendPosition = NA_integer_, ...)
```

Arguments

`x`	The trial design, obtained from `getDesignGroupSequential`, `getDesignInverseNormal` or `getDesignFisher`.
`y`	Not available for this kind of plot (is only defined to be compatible to the generic plot function).
`main`	The main title.
`xlab`	The x-axis label.
`ylab`	The y-axis label.
`type`	The plot type (default = `1`). The following plot types are available: • `1`: creates a 'Boundary Plot' • `2`: creates an 'Average Sample Size and Power / Early Stop' plot • `3`: creates a 'Stage Levels Plot' • `4`: creates a 'Power' plot • `5`: creates a 'Stopping Probability' plot • `6`: creates an 'Error Spending Plot'
`palette`	The palette, default is `"Set1"`.
`theta`	A vector of theta values.
`nMax`	The maximum sample size.
`plotPointsEnabled`	If `TRUE`, additional points will be plotted.
`legendPosition`	The position of the legend. By default (`NA_integer_`) the algorithm tries to find a suitable position. Choose one of the following values to specify the position manually: • `-1`: no legend will be shown • `NA`: the algorithm tries to find a suitable position • `0`: legend position outside plot • `1`: legend position left top

- 2: legend position left center
- 3: legend position left bottom
- 4: legend position right top
- 5: legend position right center
- 6: legend position right bottom

... Optional ggplot2 arguments.

Details

Generic function to plot a trial design.

Generic function to plot a trial design.

See Also

TrialDesignSet_plot to compare different designs or design parameters visual.

Examples

```
design <- getDesignInverseNormal(kMax = 3, alpha = 0.025,
    typeOfDesign = "asKD", gammaA = 2,
    informationRates = c(0.2, 0.7, 1),
    typeBetaSpending = "bsOF")

if (require(ggplot2)) {
plot(design) # default: type = 1
plot(design, type = 2, nMax = 20, theta = seq(0, 1, 0.05))
plot(design, type = 2, nMax = 20, theta = seq(0, 1, 0.05))
plot(design, type = 3)
plot(design, type = 4, nMax = 20, theta = seq(0, 1, 0.05))
plot(design, type = 5, nMax = 20, theta = seq(0, 1, 0.05))
plot(design, type = 6)
}
```

TrialDesign_summary

Object Summary

Description

Displays a summary of the trial design. Applicable for group sequential design, inverse normal design and Fisher design.

Usage

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

Details

Summarizes the parameters and results of a design.

writeDataset *Write Dataset*

Description

Writes a dataset to a CSV file.

Usage

```
writeDataset(dataset, file, ..., append = FALSE, quote = TRUE, sep = ",",
  eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = NA,
  qmethod = "double", fileEncoding = "UTF-8")
```

Arguments

dataset	A dataset.
file	The target CSV file.
...	Further arguments to be passed to codewrite.table.
append	Logical. Only relevant if file is a character string. If TRUE, the output is appended to the file. If FALSE, any existing file of the name is destroyed.
quote	The set of quoting characters. To disable quoting altogether, use quote = "". See scan for the behaviour on quotes embedded in quotes. Quoting is only considered for columns read as character, which is all of them unless colClasses is specified.
sep	The field separator character. Values on each line of the file are separated by this character. If sep = "," (the default for writeDataset) the separator is a comma.
eol	The character(s) to print at the end of each line (row).
na	The string to use for missing values in the data.
dec	The character used in the file for decimal points.
row.names	Either a logical value indicating whether the row names of dataset are to be written along with dataset, or a character vector of row names to be written.
col.names	Either a logical value indicating whether the column names of dataset are to be written along with dataset, or a character vector of column names to be written. See the section on 'CSV files' for the meaning of col.names = NA.
qmethod	A character string specifying how to deal with embedded double quote characters when quoting strings. Must be one of "double" (default in writeDataset) or "escape".
fileEncoding	Character string: if non-empty declares the encoding used on a file (not a connection) so the character data can be re-encoded. See the 'Encoding' section of the help for file, the 'R Data Import/Export Manual' and 'Note'.

Details

writeDataset is a wrapper function that coerces the dataset to a data frame and uses write.table to write it to a CSV file.

See Also

- writeDatasets for writing multiple datasets,
- readDataset for reading a single dataset,
- readDatasets for reading multiple datasets.

---

writeDatasets *Write Multiple Datasets*

Description

Writes a list of datasets to a CSV file.

Usage

```
writeDatasets(datasets, file, ..., append = FALSE, quote = TRUE,
  sep = ",", eol = "\n", na = "NA", dec = ".", row.names = TRUE,
  col.names = NA, qmethod = "double", fileEncoding = "UTF-8")
```

Arguments

datasets	A list of datasets.
file	The target CSV file.
...	Further arguments to be passed to codewrite.table.
append	Logical. Only relevant if file is a character string. If TRUE, the output is appended to the file. If FALSE, any existing file of the name is destroyed.
quote	The set of quoting characters. To disable quoting altogether, use quote = "". See scan for the behaviour on quotes embedded in quotes. Quoting is only considered for columns read as character, which is all of them unless colClasses is specified.
sep	The field separator character. Values on each line of the file are separated by this character. If sep = "," (the default for writeDatasets) the separator is a comma.
eol	The character(s) to print at the end of each line (row).
na	The string to use for missing values in the data.
dec	The character used in the file for decimal points.
row.names	Either a logical value indicating whether the row names of dataset are to be written along with dataset, or a character vector of row names to be written.
col.names	Either a logical value indicating whether the column names of dataset are to be written along with dataset, or a character vector of column names to be written. See the section on 'CSV files' for the meaning of col.names = NA.
qmethod	A character string specifying how to deal with embedded double quote characters when quoting strings. Must be one of "double" (default in writeDatasets) or "escape".
fileEncoding	Character string: if non-empty declares the encoding used on a file (not a connection) so the character data can be re-encoded. See the 'Encoding' section of the help for file, the 'R Data Import/Export Manual' and 'Note'.

Details

The format of the CSV file is optimized for usage of `readDatasets`.

See Also

- `writeDataset` for writing a single dataset,
- `readDatasets` for reading multiple datasets,
- `readDataset` for reading a single dataset.

`[,TrialDesignSet-method`

Access Trial Design by Index

Description

Function to the `TrialDesign` at position `i` in a `TrialDesignSet` object.

Usage

```
## S4 method for signature 'TrialDesignSet'
x[i, j = NA_character_]
```

Details

Can be used to iterate with "[index]"-syntax over all designs in a design set.

Index