Package 'rpact'

May 31, 2019

Title Confirmatory Adaptive Clinical Trial Design and Analysis

Version 2.0.1

Date 2019-05-29

Description Design and analysis of confirmatory adaptive clinical trials with continuous, binary, and survival endpoints according to the methods described in the monograph by Wassmer and Brannath (2016) <doi:10.1007/978-3-319-32562-0>. This includes classical group sequential as well as multi-stage adaptive hypotheses tests that are based on the combination testing principle.

License GPL-3

Encoding UTF-8

LazyData true

URL https://www.rpact.org

BugReports https://bugreport.rpact.org

Language en-US

Depends R (>= 3.4.0)

Imports methods,
stats,
utils,
graphics,
tools,
Rcpp (>= 1.0.0)

LinkingTo Rcpp

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

RoxygenNote 6.1.1

Collate 'RcppExports.R'
'f_core_constants.R'
'class_core_parameter_set.R'
'class_core_plot_settings.R'
'class_analysis_dataset.R'
'f_core_plot.R'
'class_design.R'
'class_analysis_stage_results.R'
'class_analysis_results.R'

'class_time.R'
'class_design_set.R'
'f_core_assertions.R'
'f_design_utilities.R'
'class_design_plan.R'
'class_design_power_and_asn.R'
'class_event_probabilities.R'
'f_simulation_survival.R'
'class_simulation_results.R'
'f_analysis.R'
'f_analysis_means.R'
'f_analysis_rates.R'
'f_analysis_survival.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'
'f_simulation_means.R'
'f_simulation_rates.R'
'pkgname.R'

R topics documented:

`AccrualTime` *Accrual Time*

Description

Class for definition of accrual time and accrual intensity.

Details

`AccrualTime` is a class for definition of accrual time and accrual intensity.

`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.

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.

EventProbabilities *Event Probabilities*

Description

Class for definition of event probabilities.

Details

`EventProbabilities` is a class for definition of event probabilities.

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.

FrameSet_as.matrix *Coerce Frame Set to a Matrix*

Description

Returns the `FrameSet` as matrix.

Usage

```
## S3 method for class 'FieldSet'
as.matrix(x, rownames.force = NA, ...)
```

Details

Coerces the frame set to a matrix.

`getAccrualTime` *Get Accrual Time*

Description

Returns a `AccrualTime` object that contains the accrual time and the accrual intensity.

Usage

```
getAccrualTime(accrualTime = NA_real_, ...,
  accrualIntensity = NA_real_, maxNumberOfSubjects = NA_real_)
```

Arguments

`accrualTime`	The assumed accrual time for the study, default is `c(0,12)` (see details).
`...`	Ensures that all arguments after `accrualTime` are be named and that a warning will be displayed if unknown arguments are passed.
`accrualIntensity`	A vector of accrual intensities, default is the relative intensity `0.1` (see details).
`maxNumberOfSubjects`	The maximum number of subjects.

Details

`accrualTime` can also be used to define a non-constant accrual over time. For this, `accrualTime` needs to be a vector that defines the accrual intervals and `accrualIntensity` needs to be specified. The first element of `accrualTime` must be equal to 0.
`accrualTime` can also be a list that combines the definition of the accrual time and accrual intensity `accrualIntensity` (see below and examples for details). If the length of `accrualTime` and the length of `accrualIntensity` are the same (i.e., the end of accrual is undefined), `maxNumberOfPatients > 0` needs to be specified and the end of accrual is calculated.

`accrualIntensity` needs to be defined if a vector of `accrualTime` is specified.
If the length of `accrualTime` and the length of `accrualIntensity` are the same (i.e., the end of accrual is undefined), `maxNumberOfPatients > 0` needs to be specified and the end of accrual is calculated. In that case, `accrualIntensity` is given by the number of subjects per time unit.
If the length of `accrualTime` equals the length of `accrualIntensity - 1` (i.e., the end of accrual is defined), `maxNumberOfPatients` is calculated. In that case, `accrualIntensity` defines the intensity how subjects enter the trial. For example, `accrualIntensity = c(1,2)` specifies that in the second accrual interval the intensity is doubled as compared to the first accrual interval. The actual accrual intensity is calculated for the calculated `maxNumberOfPatients`.

Value

Returns a `AccrualTime` object.

Examples

```
# Case 1

# > End of accrual, absolute accrual intensity and `maxNumberOfSubjects` are given,
# > `followUpTime`** shall be calculated.

## Example: vector based definition

accrualTime <- getAccrualTime(accrualTime = c(0, 6, 30),
    accrualIntensity = c(22, 33), maxNumberOfSubjects = 924)
accrualTime


## Example: list based definition

accrualTime <- getAccrualTime(list(
    "0 - <6"   = 22,
    "6 - <=30" = 33),
    maxNumberOfSubjects = 924)
accrualTime


## Example: how to use accrual time object

getSampleSizeSurvival(accrualTime = accrualTime, pi1 = 0.4, pi2 = 0.2)


# Case 2

# > End of accrual, relative accrual intensity and `maxNumberOfSubjects` are given,
# > absolute accrual intensity* and `followUpTime`** shall be calculated.

## Example: vector based definition

accrualTime <- getAccrualTime(accrualTime = c(0, 6, 30),
    accrualIntensity = c(0.22, 0.33), maxNumberOfSubjects = 1000)
accrualTime


## Example: list based definition

accrualTime <- getAccrualTime(list(
    "0 - <6"   = 0.22,
    "6 - <=30" = 0.33),
    maxNumberOfSubjects = 1000)
accrualTime


## Example: how to use accrual time object

getSampleSizeSurvival(accrualTime = accrualTime, pi1 = 0.4, pi2 = 0.2)
```

```
# Case 3

# > End of accrual and absolute accrual intensity are given,
# > `maxNumberOfSubjects`* and `followUpTime`** shall be calculated.

## Example: vector based definition

accrualTime <- getAccrualTime(accrualTime = c(0, 6, 30), accrualIntensity = c(22, 33))


## Example: list based definition

accrualTime <- getAccrualTime(list(
    "0 - <6"   = 22,
    "6 - <=30" = 33))
accrualTime


## Example: how to use accrual time object

getSampleSizeSurvival(accrualTime = accrualTime, pi1 = 0.4, pi2 = 0.2)


# Case 4

# > End of accrual, relative accrual intensity and `followUpTime` are given,
# > absolute accrual intensity** and `maxNumberOfSubjects`** shall be calculated.

## Example: vector based definition

accrualTime <- getAccrualTime(accrualTime = c(0, 6, 30), accrualIntensity = c(0.22, 0.33)
accrualTime


## Example: list based definition

accrualTime <- getAccrualTime(list(
    "0 - <6"   = 0.22,
    "6 - <=30" = 0.33))
accrualTime


## Example: how to use accrual time object

getSampleSizeSurvival(accrualTime = accrualTime, pi1 = 0.4, pi2 = 0.2)


# Case 5

# > `maxNumberOfSubjects` and absolute accrual intensity are given,
# > absolute accrual intensity*, end of accrual* and `followUpTime`** shall be calculatec

## Example: vector based definition

accrualTime <- getAccrualTime(accrualTime = c(0, 6),
    accrualIntensity = c(22, 33), maxNumberOfSubjects = 1000)
accrualTime
```

```
## Example: list based definition

accrualTime <- getAccrualTime(list(
    "0 - <6" = 22,
    "6"      = 33),
    maxNumberOfSubjects = 1000)
accrualTime


## Example: how to use accrual time object

getSampleSizeSurvival(accrualTime = accrualTime, pi1 = 0.4, pi2 = 0.2)


# Case 6 (not possible)

# > `maxNumberOfSubjects` and relative accrual intensity are given,
# > absolute accrual intensity[x], end of accrual* and `followUpTime`** shall be calculat

## Example: vector based definition

accrualTime <- getAccrualTime(accrualTime = c(0, 6),
    accrualIntensity = c(0.22, 0.33), maxNumberOfSubjects = 1000)
accrualTime


## Example: list based definition

accrualTime <- getAccrualTime(list(
    "0 - <6" = 0.22,
    "6"      = 0.33),
    maxNumberOfSubjects = 1000)
accrualTime


## Example: how to use accrual time object

# Case 6 is not allowed and therefore an error will be shown:

tryCatch({
    getSampleSizeSurvival(accrualTime = accrualTime, pi1 = 0.4, pi2 = 0.2)
}, error = function(e) {
    print(e$message)
})


# Case 7

# > `followUpTime` and absolute accrual intensity are given,
# > end of accrual** and `maxNumberOfSubjects`** shall be calculated

## Example: vector based definition

accrualTime <- getAccrualTime(accrualTime = c(0, 6), accrualIntensity = c(22, 33))
accrualTime
```

```
## Example: list based definition

accrualTime <- getAccrualTime(list(
    "0 - <6" = 22,
    "6"      = 33))
accrualTime


## Example: how to use accrual time object

getSampleSizeSurvival(accrualTime = accrualTime,
    pi1 = 0.4, pi2 = 0.2, followUpTime = 6)


# Case 8 (not possible)

# > `followUpTime` and relative accrual intensity are given,
# > absolute accrual intensity[x], end of accrual and `maxNumberOfSubjects` shall be calc

## Example: vector based definition

accrualTime <- getAccrualTime(accrualTime = c(0, 6), accrualIntensity = c(0.22, 0.33))
accrualTime


## Example: list based definition

accrualTime <- getAccrualTime(list(
    "0 - <6" = 0.22,
    "6"      = 0.33))
accrualTime


## Example: how to use accrual time object

# Case 8 is not allowed and therefore an error will be shown:

tryCatch({
    getSampleSizeSurvival(accrualTime = accrualTime, pi1 = 0.4, pi2 = 0.2, followUpTime =
}, error = function(e) {
    print(e$message)
})


# How to show accrual time details

# You can use a sample size or power object as argument for function `getAccrualTime`:

sampleSize <- getSampleSizeSurvival(accrualTime = c(0, 6), accrualIntensity = c(22, 53),
    lambda2 = 0.05, hazardRatio = 0.8, followUpTime = 6)
sampleSize
accrualTime <- getAccrualTime(sampleSize)
accrualTime
```

`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 non-inferiority designs, this is the non-inferiority 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)
```

`getAvailablePlotTypes` *Get Available Plot Types*

Description

Function to identify the available plot types of an object.

Usage

```
getAvailablePlotTypes(obj, output = c("numeric", "caption", "numcap",
  "capnum"), numberInCaptionEnabled = FALSE)
```

Arguments

`obj`	The object for which the plot types shall be identified, e.g. produced by `getDesignGroupSequen` or `getSampleSizeMeans`.
`output`	The output type. Can be one of `c("numeric", "caption", "numcap", "capnum")`.
`numberInCaptionEnabled`	If `TRUE`, the number will be added to the caption, default is `FALSE`.

Details

`output`:

1. `numeric`: numeric output
2. `caption`: caption as character output
3. `numcap`: list with number and caption
4. `capnum`: list with caption and number

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 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) needs to 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

plot.StageResults or plot.AnalysisResults 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
```

getData *Get Simulation Data*

Description

Returns the aggregated simulation data.

Usage

```
getData(x)
```

Arguments

x	An SimulationResults object created by getSimulationMeans, getSimulationRate or getSimulationSurvival.

Details

This data are the base for creation of the small statistics in the simulation results output.

---

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 = NA,
  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 = 1, 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 = NA,
  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, 3,..., 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, defined on the test statistic z scale (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 ("asOF"), 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. Vector of length kMax containing the cumulative alpha-spending up to each interim stage.

userBetaSpending
: The user defined beta spending. Vector of length kMax containing the cumulative beta-spending up to each interim stage.

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()

# 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")
```

`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, 3,..., 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 ("asOF"), 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. Vector of length kMax containing the cumulative alpha-spending up to each interim stage.

`userBetaSpending`
The user defined beta spending. Vector of length kMax containing the cumulative beta-spending up to each interim stage.

`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()

# 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")
```

`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)
```

`getEventProbabilities` *Get Event Probabilities*

Description

Returns the event probabilities for specified parameters at given time vector.

Usage

```
getEventProbabilities(time, ..., accrualTime = C_ACCRUAL_TIME_DEFAULT,
  accrualIntensity = C_ACCRUAL_INTENSITY_DEFAULT, kappa = 1,
  piecewiseSurvivalTime = NA_real_, lambda2 = NA_real_,
  lambda1 = NA_real_, allocationRatioPlanned = 1,
  hazardRatio = NA_real_, dropoutRate1 = C_DROP_OUT_RATE_1_DEFAULT,
  dropoutRate2 = C_DROP_OUT_RATE_2_DEFAULT,
  dropoutTime = C_DROP_OUT_TIME_DEFAULT,
  maxNumberOfSubjects = NA_real_)
```

Arguments

`time`	A numeric vector with time values.
`...`	Ensures that all arguments are be named and that a warning will be displayed if unknown arguments are passed.
`accrualTime`	The assumed accrual time intervals for the study, default is `c(0, 12)` (see details).
`accrualIntensity`	A vector of accrual intensities, default is the relative intensity `0.1` (see details).
`kappa`	The shape parameter of the Weibull distribution, default is `1`. The Weibull distribution cannot be used for the piecewise definition of the survival time distribution. Note that the parameters `shape` and `scale` in `Weibull` are equivalent to `kappa` and `1 / lambda`, respectively, in rpact.
`piecewiseSurvivalTime`	A vector that specifies the time intervals for the piecewise definition of the exponential survival time cumulative distribution function (see details).
`lambda2`	The assumed hazard rate in the reference group, there is no default. lambda2 can also be used to define piecewise exponentially distributed survival times (see details).
`lambda1`	The assumed hazard rate in the treatment group, there is no default. lambda1 can also be used to define piecewise exponentially distributed survival times (see details).
`allocationRatioPlanned`	The planned allocation ratio, default is `1`. If `allocationRatioPlanned = 0` is entered, the optimal allocation ratio yielding the smallest number of subjects is determined.
`hazardRatio`	The vector of hazard ratios under consideration. If the event or hazard rates in both treatment groups are defined, the hazard ratio needs not to be specified as it is calculated.

dropoutRate1 The assumed drop-out rate in the treatment group, default is 0.

dropoutRate2 The assumed drop-out rate in the control group, default is 0.

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

maxNumberOfSubjects
If maxNumberOfSubjects > 0 is specified, the end of accrual at specified accrualIntensity for the specified number of subjects is determined or accrualIntensity is calculated at fixed end of accrual.

Details

For details of the parameters see getSampleSizeSurvival.

Value

Returns a EventProbabilities object.

---

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 non-inferiority designs, this is the non-inferiority 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.

getLogLevel *Get Log Level*

Description

Returns the current rpact log level.

Usage

```
getLogLevel()
```

Details

This function is intended for debugging purposes only.

Examples

```
## Not run:
getLogLevel()

## End(Not run)
```

`getNumberOfSubjects`

Get Number Of Subjects

Description

Returns the number of recruited subjects at given time vector.

Usage

```
getNumberOfSubjects(time, ..., accrualTime = C_ACCRUAL_TIME_DEFAULT,
  accrualIntensity = C_ACCRUAL_INTENSITY_DEFAULT,
  maxNumberOfSubjects = NA_real_)
```

Arguments

`time`	A numeric vector with time values.
`...`	Ensures that all arguments are be named and that a warning will be displayed if unknown arguments are passed.
`accrualTime`	The assumed accrual time intervals for the study, default is `c(0,12)` (see details).
`accrualIntensity`	A vector of accrual intensities, default is the relative intensity `0.1` (see details).
`maxNumberOfSubjects`	If `maxNumberOfSubjects > 0` is specified, the end of accrual at specified `accrualIntensity` for the specified number of subjects is determined or `accrualIntensity` is calculated at fixed end of accrual.

Details

For details of the parameters `accrualTime` and `accrualIntensity` see `getSampleSizeSurvival`.

Value

Returns a `NumberOfSubjects` object.

`getPiecewiseSurvivalTime`

Get Piecewise Survival Time

Description

Returns a `PiecewiseSurvivalTime` object that contains the all relevant parameters of an exponential survival time cumulative distribution function.

Usage

```
getPiecewiseSurvivalTime(piecewiseSurvivalTime = NA_real_, ...,
  lambda1 = NA_real_, lambda2 = NA_real_, hazardRatio = NA_real_,
  pi1 = NA_real_, pi2 = NA_real_, eventTime = C_EVENT_TIME_DEFAULT,
  kappa = 1, delayedResponseAllowed = FALSE)
```

Arguments

`piecewiseSurvivalTime`
: A vector that specifies the time intervals for the piecewise definition of the exponential survival time cumulative distribution function (see details).

`...`
: Ensures that all arguments after `piecewiseSurvivalTime` are be named and that a warning will be displayed if unknown arguments are passed.

`lambda1`
: The assumed hazard rate in the treatment group, there is no default. lambda1 can also be used to define piecewise exponentially distributed survival times (see details).

`lambda2`
: The assumed hazard rate in the reference group, there is no default. lambda2 can also be used to define piecewise exponentially distributed survival times (see details).

`hazardRatio`
: The vector of hazard ratios under consideration. If the event or hazard rates in both treatment groups are defined, the hazard ratio needs not to be specified as it is calculated.

`pi1`
: The assumed event rate in the treatment group, default is `seq(0.4, 0.6, 0.1)`.

`pi2`
: The assumed event rate in the control group, default is 0.2.

`eventTime`
: The assumed time under which the event rates are calculated, default is `12`.

`kappa`
: The shape parameter of the Weibull distribution, default is `1`. The Weibull distribution cannot be used for the piecewise definition of the survival time distribution. Note that the parameters `shape` and `scale` in `Weibull` are equivalent to `kappa` and `1 / lambda`, respectively, in rpact.

`delayedResponseAllowed`
: If `TRUE`, delayed response is allowed; otherwise it will be validatet that the definition is not delayed, default is `FALSE`.

Details

`piecewiseSurvivalTime` The first element of this vector must be equal to `0`. `piecewiseSurvivalTime` can also be a list that combines the definition of the time intervals and hazard rates in the reference group. The definition of the survival time in the treatment group is obtained by the specification of the hazard ratio (see examples for details).

Value

Returns a `PiecewiseSurvivalTime` object.

Examples

```
pwst <- getPiecewiseSurvivalTime(lambda2 = 0.5, hazardRatio = 0.8)
pwst

pwst <- getPiecewiseSurvivalTime(lambda2 = 0.5, lambda1 = 0.4)
pwst

pwst <- getPiecewiseSurvivalTime(pi2 = 0.5, hazardRatio = 0.8)
pwst

pwst <- getPiecewiseSurvivalTime(pi2 = 0.5, pi1 = 0.4)
pwst
```

```
pwst <- getPiecewiseSurvivalTime(pi1 = 0.3)
pwst

pwst <- getPiecewiseSurvivalTime(hazardRatio = c(0.6, 0.8), lambda2 = 0.4)
pwst

pwst <- getPiecewiseSurvivalTime(piecewiseSurvivalTime = c(0, 6, 9),
    lambda2 = c(0.025, 0.04, 0.015), hazardRatio = 0.8)
pwst

pwst <- getPiecewiseSurvivalTime(piecewiseSurvivalTime = c(0, 6, 9),
    lambda2 = c(0.025, 0.04, 0.015),
    lambda1 = c(0.025, 0.04, 0.015) * 0.8)
pwst

pwst <- getPiecewiseSurvivalTime(list(
    "0 - <6"   = 0.025,
    "6 - <9"   = 0.04,
    "9 - <15"  = 0.015,
    "15 - <21" = 0.01,
    ">=21"     = 0.007), hazardRatio = 0.75)
pwst



# The object created by getPiecewiseSurvivalTime() can be used directly in getSampleSizeS
getSampleSizeSurvival(piecewiseSurvivalTime = pwst)

# The object created by getPiecewiseSurvivalTime() can be used directly in getPowerSurviv
getPowerSurvival(piecewiseSurvivalTime = pwst,
    maxNumberOfEvents = 40, maxNumberOfSubjects = 100)
```

`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 standardized effect sizes.
`nMax`	The maximum sample size.

Details

This function returns the power and average sample number (ASN) of the specified design for the prototype case which is testing H0: mu = mu0 in a one-sample design. theta represents the standardized effect (mu - mu0)/sigma and power and ASN is calculated for maximum sample size nMax. For other designs than the one-sample test of a mean the standardized effect needs to be adjusted accordingly.

Value

Returns a `PowerAndAverageSampleNumberResult` object.

Examples

```
getPowerAndAverageSampleNumber(
    getDesignGroupSequential(),
    theta = seq(-1, 1, 0.5), nMax = 100)
```

`getPowerMeans` *Get Power Means*

Description

Returns the power, stopping probabilities, and expected sample size for testing means in one or two samples at given sample size.

Usage

```
getPowerMeans(design = NULL, ..., normalApproximation = FALSE,
  meanRatio = FALSE, thetaH0 = ifelse(meanRatio, 1, 0),
  alternative = C_ALTERNATIVE_POWER_SIMULATION_DEFAULT,
  stDev = C_STDEV_DEFAULT, directionUpper = NA,
  maxNumberOfSubjects = NA_real_, groups = 2,
  allocationRatioPlanned = NA_real_)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, `alpha`, `beta`, `twoSidedPower`, and `sided` can be directly entered as argument.
`...`	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, i.e., the calculations are performed with the t distribution.
`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` or `1` for difference and ratio testing, respectively.

`alternative` The alternative hypothesis value. This can be a vector of assumed alternatives, default is `seq(0,1,0.2)`.

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

`directionUpper` Specifies the direction of the alternative, only applicable for one-sided testing, default is `TRUE`.

`maxNumberOfSubjects` `maxNumberOfSubjects > 0` needs to be specified. If accrual time and accrual intensity is specified, this will be calculated.

`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`.

Details

At given design the function calculates the power, stopping probabilities, and expected sample size, for testing means at given sample size. In a two treatment groups design, additionally, an allocation ratio = n1/n2 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. For the specified sample size, critical bounds and stopping for futility bounds are provided at the effect scale (mean, mean difference, or mean ratio, respectively)

Value

Returns a `TrialDesignPlanMeans` object.

Examples

```
# Calculate the power, stopping probabilities, and expected sample size for testing H0:
# mu1 - mu2 = 0 in a two-armed design
# against a range of alternatives H1: mu1 - m2 = delta, delta = (0, 1, 2, 3, 4, 5),
# standard deviation sigma = 8, maximum sample size N = 80 (both treatment arms),
# and an allocation ratio n1/n2 = 2. The design is a three stage O'Brien & Fleming design
# with non-binding futility bounds (-0.5, 0.5) for the two interims.
# The computation takes into account that the t test is used (normalApproximation = FALSE
getPowerMeans(getDesignGroupSequential(alpha = 0.025,
    sided = 1, futilityBounds = c(-0.5, 0.5)),
    groups = 2, alternative = c(0:5), stDev = 8,
    normalApproximation = FALSE, maxNumberOfSubjects = 80,
    allocationRatioPlanned = 2)
```

getPowerRates *Get Power Rates*

Description

Returns the power, stopping probabilities, and expected sample size for testing rates in one or two samples at given sample sizes.

Usage

```
getPowerRates(design = NULL, ..., normalApproximation = TRUE,
  riskRatio = FALSE, thetaH0 = ifelse(riskRatio, 1, 0),
  pi1 = C_PI_1_DEFAULT, pi2 = 0.2, directionUpper = NA,
  maxNumberOfSubjects = NA_real_, groups = 2,
  allocationRatioPlanned = NA_real_)
```

Arguments

design	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, alpha, beta, and sided can be directly entered as argument
...	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 or 1 for difference and ratio testing, respectively.
pi1	The assumed probability in the active treatment group if two treatment groups are considered, or the alternative probability for a one treatment group design, default is seq(0.2,0.5,0.1).
pi2	The assumed probability in the reference group if two treatment groups are considered, default is 0.2.
directionUpper	Specifies the direction of the alternative, only applicable for one-sided testing, default is TRUE.
maxNumberOfSubjects	maxNumberOfSubjects > 0 needs to be specified. If accrual time and accrual intensity is specified, this will be calculated.
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.

Details

At given design the function calculates the power, stopping probabilities, and expected sample size, for testing rates for given maximum sample size. The sample sizes over the stages are calculated according to the specified information rate in the design. In a two treatment groups design, additionally, an allocation ratio = n1/n2 can be specified. If a null hypothesis value thetaH0 != 0 for testing the difference of two rates or thetaH0 != 1 for testing the risk ratio is specified, the formulas according to Farrington & Manning (Statistics in Medicine, 1990) are used (only one-sided testing). Critical bounds and stopping for futility bounds are provided at the effect scale (rate, rate difference, or rate ratio, respectively). For the two-sample case, the calculation here is performed at fixed pi2 as given as argument in the function.

Value

Returns a `TrialDesignPlanRates` object.

Examples

```
# Calculate the power, stopping probabilities, and expected sample size in a two-armed
# design at given maximum sample size N = 200
# in a three-stage O'Brien & Fleming design with information rate vector (0.2,0.5,1),
# non-binding futility boundaries (0,0), i.e.,
# the study stops for futility if the p-value exceeds 0.5 at interm, and
# allocation ratio = 2 for a range of pi1 values when testing H0: pi1 - pi2 = -0.1:
getPowerRates(getDesignGroupSequential(informationRates = c(0.2,0.5,1),
    futilityBounds = c(0,0)), groups = 2, thetaH0 = -0.1,
    pi1 = seq(0.3, 0.6, 0.1), directionUpper = FALSE,
    pi2 = 0.7, allocationRatioPlanned = 2, maxNumberOfSubjects = 200)

# Calculate the power, stopping probabilities, and expected sample size in a single
# arm design at given maximum sample size N = 60 in a three-stage two-sided
# O'Brien & Fleming design with information rate vector (0.2,0.5,1)
# for a range of pi1 values when testing H0: pi = 0.3:
getPowerRates(getDesignGroupSequential(informationRates = c(0.2,0.5,1),
    sided = 2), groups = 1, thetaH0 = 0.3, pi1 = seq(0.3, 0.5, 0.05),
    maxNumberOfSubjects = 60)
```

---

getPowerSurvival *Get Power Survival*

Description

Returns the power, stopping probabilities, and expected sample size for testing the hazard ratio in a two treatment groups survival design.

Usage

```
getPowerSurvival(design = NULL, ...,
  typeOfComputation = c("Schoenfeld", "Freedman", "HsiehFreedman"),
  thetaH0 = C_THETA_H0_SURVIVAL_DEFAULT, directionUpper = NA,
  pi1 = NA_real_, pi2 = NA_real_, lambda1 = NA_real_,
```

```
  lambda2 = NA_real_, kappa = 1, hazardRatio = NA_real_,
  piecewiseSurvivalTime = NA_real_, allocationRatioPlanned = 1,
  eventTime = C_EVENT_TIME_DEFAULT,
  accrualTime = C_ACCRUAL_TIME_DEFAULT,
  accrualIntensity = C_ACCRUAL_INTENSITY_DEFAULT,
  maxNumberOfSubjects = NA_real_, maxNumberOfEvents = NA_real_,
  dropoutRate1 = C_DROP_OUT_RATE_1_DEFAULT,
  dropoutRate2 = C_DROP_OUT_RATE_2_DEFAULT,
  dropoutTime = C_DROP_OUT_TIME_DEFAULT)
```

Arguments

design	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, alpha, beta, twoSidedPower, and sided can be directly entered as argument.
...	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). For non-inferiority testing (i.e., thetaH0 != 1), only Schoenfelds formula can be used
thetaH0	The null hypothesis value. The default value is 1. For one-sided testing, a bound for testing H0: hazard ratio = thetaH0 != 1 can be specified.
directionUpper	Specifies the direction of the alternative, only applicable for one-sided testing, default is TRUE.
pi1	The assumed event rate in the treatment group, default is seq(0.2,0.5,0.1).
pi2	The assumed event rate in the control group, default is 0.2.
lambda1	The assumed hazard rate in the treatment group, there is no default. lambda1 can also be used to define piecewise exponentially distributed survival times (see details).
lambda2	The assumed hazard rate in the reference group, there is no default. lambda2 can also be used to define piecewise exponentially distributed survival times (see details).
kappa	The shape parameter of the Weibull distribution, default is 1. The Weibull distribution cannot be used for the piecewise definition of the survival time distribution. Note that the parameters shape and scale in Weibull are equivalent to kappa and 1 / lambda, respectively, in rpact.
hazardRatio	The vector of hazard ratios under consideration. If the event or hazard rates in both treatment groups are defined, the hazard ratio needs not to be specified as it is calculated.
piecewiseSurvivalTime	A vector that specifies the time intervals for the piecewise definition of the exponential survival time cumulative distribution function (see details).
allocationRatioPlanned	The planned allocation ratio, default is 1.
eventTime	The assumed time under which the event rates are calculated, default is 12.
accrualTime	The assumed accrual time intervals for the study, default is c(0,12) (see details).

accrualIntensity
: A vector of accrual intensities, default is 1 (see details).

maxNumberOfSubjects
: maxNumberOfSubjects > 0 needs to be specified. If accrual time and accrual intensity is specified, this will be calculated.

maxNumberOfEvents
: maxNumberOfEvents > 0 is the maximum number of events, determines the power of the test and needs to be specified.

dropoutRate1
: The assumed drop-out rate in the treatment group, default is 0.

dropoutRate2
: The assumed drop-out rate in the control 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 power, stopping probabilities, and expected sample size at given number of events and number of subjects. It also calculates the time when the required events are expected under the given assumptions (exponentially, piecewise exponentially, or Weibull distributed survival times and constant or non-constant piecewise accrual). Additionally, an allocation ratio = n1/n2 can be specified where n1 and n2 are the number of subjects in the two treatment groups.

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). These formulas are generalized to piecewise survival times and non-constant piecewise accrual over time.

piecewiseSurvivalTime The first element of this vector must be equal to 0. piecewiseSurvivalTime can also be a list that combines the definition of the time intervals and hazard rates in the reference group. The definition of the survival time in the treatment group is obtained by the specification of the hazard ratio (see examples for details).

accrualTime can also be used to define a non-constant accrual over time. For this, accrualTime needs to be a vector that defines the accrual intervals and accrualIntensity needs to be specified. The first element of accrualTime must be equal to 0.
accrualTime can also be a list that combines the definition of the accrual time and accrual intensity accrualIntensity (see below and examples for details). If the length of accrualTime and the length of accrualIntensity are the same (i.e., the end of accrual is undefined), maxNumberOfPatients > 0 needs to be specified and the end of accrual is calculated.

accrualIntensity needs to be defined if a vector of accrualTime is specified.
If the length of accrualTime and the length of accrualIntensity are the same (i.e., the end of accrual is undefined), maxNumberOfPatients > 0 needs to be specified and the end of accrual is calculated. In that case, accrualIntensity is given by the number of subjects per time unit.
If the length of accrualTime equals the length of accrualIntensity - 1 (i.e., the end of accrual is defined), maxNumberOfPatients is calculated. In that case, accrualIntensity defines the intensity how subjects enter the trial. For example, accrualIntensity = c(1,2) specifies that in the second accrual interval the intensity is doubled as compared to the first accrual interval. The actual accrual intensity is calculated for the calculated maxNumberOfPatients.

Value

Returns a TrialDesignPlanSurvival object.

Examples

```
# Fixed sample size with minimum required definitions, pi1 = c(0.4,0.5,0.5) and
# pi2 = 0.2 at event time 12, accrual time 12 and follow-up time 6 as default
getPowerSurvival(maxNumberOfEvents = 40, maxNumberOfSubjects = 200)



# Four stage O'Brien & Fleming group sequential design with minimum required
# definitions, pi1 = c(0.4,0.5,0.5) and pi2 = 0.2 at event time 12,
# accrual time 12 and follow-up time 6 as default
getPowerSurvival(design = getDesignGroupSequential(kMax = 4),
    maxNumberOfEvents = 40, maxNumberOfSubjects = 200)

# For fixed sample design, determine necessary accrual time if 200 subjects and
# 30 subjects per time unit can be recruited
getPowerSurvival(maxNumberOfEvents = 40, accrualTime = c(0),
    accrualIntensity = 30, maxNumberOfSubjects = 200)

# Determine necessary accrual time if 200 subjects and if the first 6 time units
# 20 subjects per time unit can be recruited, then 30 subjects per time unit
getPowerSurvival(maxNumberOfEvents = 40, accrualTime = c(0, 6),
    accrualIntensity = c(20, 30), maxNumberOfSubjects = 200)

# Determine maximum number of Subjects if the first 6 time units 20 subjects per
# time unit can be recruited, and after 10 time units 30 subjects per time unit
getPowerSurvival(maxNumberOfEvents = 40, accrualTime = c(0, 6, 10), accrualIntensity = c(

# Specify accrual time as a list
at <- list(
    "0 - <6"  = 20,
    "6 - Inf" = 30)
getPowerSurvival(maxNumberOfEvents = 40, accrualTime = at, maxNumberOfSubjects = 200)

# Specify accrual time as a list, if maximum number of subjects need to be calculated
at <- list(
    "0 - <6"   = 20,
    "6 - <=10" = 30)
getPowerSurvival(maxNumberOfEvents = 40, accrualTime = at)

# Specify effect size for a two-stage group design with O'Brien & Fleming boundaries
# Effect size is based on event rates at specified event time, directionUpper = FALSE
# needs to be specified because it should be shown that hazard ratio < 1
getPowerSurvival(design = getDesignGroupSequential(kMax = 2), pi1 = 0.2, pi2 = 0.3,
    eventTime = 24, maxNumberOfEvents = 40, maxNumberOfSubjects = 200, directionUpper = F

# Effect size is based on event rate at specified event time for the reference group
# and hazard ratio, directionUpper = FALSE needs to be specified
# because it should be shown that hazard ratio < 1
getPowerSurvival(design = getDesignGroupSequential(kMax = 2), hazardRatio = 0.5, pi2 = 0.
    eventTime = 24, maxNumberOfEvents = 40, maxNumberOfSubjects = 200, directionUpper = F

# Effect size is based on hazard rate for the reference group and hazard ratio,
# directionUpper = FALSE needs to be specified because it should be shown that hazard rat
getPowerSurvival(design = getDesignGroupSequential(kMax = 2), hazardRatio = 0.5,
    lambda2 = 0.02, maxNumberOfEvents = 40, maxNumberOfSubjects = 200, directionUpper = F
```

```
# Specification of piecewise exponential survival time and hazard ratios
getPowerSurvival(design = getDesignGroupSequential(kMax = 2),
    piecewiseSurvivalTime = c(0, 5, 10), lambda2 = c(0.01,0.02,0.04),
    hazardRatio = c(1.5, 1.8, 2),  maxNumberOfEvents = 40, maxNumberOfSubjects = 200)

# Specification of piecewise exponential survival time as list and hazard ratios
pws <- list(
    "0 - <5"  = 0.01,
   "5 - <10" = 0.02,
   ">=10"    = 0.04)
getPowerSurvival(design = getDesignGroupSequential(kMax = 2),
   piecewiseSurvivalTime = pws, hazardRatio = c(1.5, 1.8, 2),
   maxNumberOfEvents = 40, maxNumberOfSubjects = 200)

# Specification of piecewise exponential survival time for both treatment arms
getPowerSurvival(design = getDesignGroupSequential(kMax = 2),
   piecewiseSurvivalTime = c(0, 5, 10), lambda2 = c(0.01, 0.02, 0.04),
   lambda1 = c(0.015,0.03,0.06),  maxNumberOfEvents = 40, maxNumberOfSubjects = 200)

# Specification of piecewise exponential survival time as a list
pws <- list(
    "0 - <5"  = 0.01,
   "5 - <10" = 0.02,
   ">=10"    = 0.04)
getPowerSurvival(design = getDesignGroupSequential(kMax = 2),
   piecewiseSurvivalTime = pws, hazardRatio = c(1.5, 1.8, 2),
   maxNumberOfEvents = 40, maxNumberOfSubjects = 200)

# Specify effect size based on median survival times
median1 <- 5
median2 <- 3
getPowerSurvival(lambda1 = log(2) / median1, lambda2 = log(2) / median2,
   maxNumberOfEvents = 40, maxNumberOfSubjects = 200, directionUpper = FALSE)

# Specify effect size based on median survival times of Weibull distribtion with kappa =
median1 <- 5
median2 <- 3
kappa <- 2
getPowerSurvival(lambda1 = log(2)^(1 / kappa) / median1,
   lambda2 = log(2)^(1 / kappa) / median2, kappa = kappa,
   maxNumberOfEvents = 40, maxNumberOfSubjects = 200, directionUpper = FALSE)
```

`getRawData` *Get Simulation Raw Data*

Description

Returns the raw data which was generated randomly for simulation.

Usage

```
getRawData(x, aggregate = FALSE)
```

Arguments

x	An SimulationResults object created by getSimulationSurvival.
aggregate	If TRUE the raw data will be aggregated similar to the result of getData, default is FALSE.

Details

This function works only if getSimulationSurvival was called with a maxNumberOfRawDatasetsPerStag (default is 0).

---

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 or two samples.

Usage

```
getSampleSizeMeans(design = NULL, ..., normalApproximation = FALSE,
  meanRatio = FALSE, thetaH0 = ifelse(meanRatio, 1, 0),
  alternative = C_ALTERNATIVE_DEFAULT, stDev = C_STDEV_DEFAULT,
  groups = 2, allocationRatioPlanned = NA_real_)
```

Arguments

design	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, alpha, beta, twoSidedPower, and sided can be directly entered as argument.
...	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, i.e., the calculations are performed with the t distribution.

`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` or `1` for difference and ratio testing, respectively.

`alternative`
: The alternative hypothesis value. This can be a vector of assumed alternatives, default is `seq(0.2,1,0.2)`.

`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 stage-wise (non-cumulated) and maximum sample size for testing means. In a two treatment groups design, additionally, an allocation ratio = n1/n2 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. Critical bounds and stopping for futility bounds are provided at the effect scale (mean, mean difference, or mean ratio, respectively) for each sample size calculation separately.

Value

Returns a `TrialDesignPlanMeans` object.

Examples

```
# Calculate sample sizes in a fixed sample size parallel group design
# with allocation ratio n1/n2 = 2 for a range of alternative values 1,...,5
# with assumed standard deviation = 3.5; two-sided alpha = 0.05, power 1 - beta = 90%:
getSampleSizeMeans(alpha = 0.05, beta = 0.1, sided = 2, groups = 2,
    alternative = seq(1, 5, 1), stDev = 3.5, allocationRatioPlanned = 2)

# Calculate sample sizes in a three-stage Pocock paired comparison design testing
# H0: mu = 2 for a range of alternative values 3,4,5 with assumed standard
# deviation = 3.5; one-sided alpha = 0.05, power 1 - beta = 90%:
getSampleSizeMeans(getDesignGroupSequential(typeOfDesign = "P", alpha = 0.05,
    sided = 1, beta = 0.1), groups = 1, thetaH0 = 2,
    alternative = seq(3, 5, 1), stDev = 3.5)
```

getSampleSizeRates *Get Sample Size Rates*

Description

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

Usage

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

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, `alpha`, `beta`, `twoSidedPower`, and `sided` can be directly entered as argument.
`...`	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` or `1` for difference and ratio testing, respectively.
`pi1`	The assumed probability in the active 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 reference 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 stage-wise (non-cumulated) and maximum sample size for testing rates. In a two treatment groups design, additionally, an allocation ratio = n1/n2 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. Critical bounds and stopping for futility bounds are provided at the effect scale (rate, rate difference, or rate ratio, respectively) for each sample size calculation separately. For the two-sample case, the calculation here is performed at fixed pi2 as given as argument in the function.

Value

Returns a TrialDesignPlanRates object.

Examples

```
# Calculate the stage-wise sample sizes, maximum sample sizes, and the optimum
# allocation ratios for a range of pi1 values when testing
# H0: pi1 - pi2 = -0.1 within a two-stage O'Brien & Fleming design;
# alpha = 0.05 one-sided, power 1- beta = 90%:
getSampleSizeRates(design = getDesignGroupSequential(kMax = 2, alpha = 0.05, beta = 0.1,
    sided = 1), groups = 2, thetaH0 = -0.1, pi1 = seq(0.4, 0.55, 0.025),
    pi2 = 0.4, allocationRatioPlanned = 0)

# Calculate the stage-wise sample sizes, maximum sample sizes, and the optimum
# allocation ratios for a range of pi1 values when testing
# H0: pi1 / pi2 = 0.80 within a three-stage O'Brien & Fleming design;
# alpha = 0.025 one-sided, power 1- beta = 90%:
getSampleSizeRates(getDesignGroupSequential(kMax = 3, alpha = 0.025, beta = 0.1,
    sided = 1), groups = 2, riskRatio = TRUE, thetaH0 = 0.80, pi1 = seq(0.3,0.5,0.025),
    pi2 = 0.3, allocationRatioPlanned = 0)
```

---

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 = NULL, ...,
  typeOfComputation = c("Schoenfeld", "Freedman", "HsiehFreedman"),
  thetaH0 = C_THETA_H0_SURVIVAL_DEFAULT, pi1 = NA_real_,
  pi2 = NA_real_, lambda1 = NA_real_, lambda2 = NA_real_,
  kappa = 1, hazardRatio = NA_real_,
  piecewiseSurvivalTime = NA_real_, allocationRatioPlanned = NA_real_,
  accountForObservationTimes = TRUE, eventTime = C_EVENT_TIME_DEFAULT,
  accrualTime = C_ACCRUAL_TIME_DEFAULT,
  accrualIntensity = C_ACCRUAL_INTENSITY_DEFAULT,
  followUpTime = NA_real_, maxNumberOfSubjects = NA_real_,
  dropoutRate1 = C_DROP_OUT_RATE_1_DEFAULT,
  dropoutRate2 = C_DROP_OUT_RATE_2_DEFAULT,
  dropoutTime = C_DROP_OUT_TIME_DEFAULT)
```

Arguments

design	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, alpha, beta, twoSidedPower, and sided can be directly entered as argument.

`...` 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). For non-inferiority testing (i.e., thetaH0 != 1), only Schoenfelds formula can be used

`thetaH0` The null hypothesis value. The default value is `1`. For one-sided testing, a bound for testing H0: hazard ratio = thetaH0 != 1 can be specified.

`pi1` The assumed event rate in the active treatment group, default is `seq(0.4,0.6,0.1)`.

`pi2` The assumed event rate in the control group, default is `0.2`.

`lambda1` The assumed hazard rate in the treatment group, there is no default. lambda1 can also be used to define piecewise exponentially distributed survival times (see details).

`lambda2` The assumed hazard rate in the reference group, there is no default. lambda2 can also be used to define piecewise exponentially distributed survival times (see details).

`kappa` The shape parameter of the Weibull distribution, default is `1`. The Weibull distribution cannot be used for the piecewise definition of the survival time distribution. Note that the parameters `shape` and `scale` in `Weibull` are equivalent to `kappa` and `1 / lambda`, respectively, in rpact.

`hazardRatio` The vector of hazard ratios under consideration. If the event or hazard rates in both treatment groups are defined, the hazard ratio needs not to be specified as it is calculated.

`piecewiseSurvivalTime`
A vector that specifies the time intervals for the piecewise definition of the exponential survival time cumulative distribution function (see details).

`allocationRatioPlanned`
The planned allocation ratio, default is `1`. If `allocationRatioPlanned = 0` is entered, the optimal allocation ratio yielding the smallest number of subjects is determined.

`accountForObservationTimes`
If `accountForObservationTimes = TRUE`, the number of subjects is calculated assuming specific accrual and follow-up time, default is `TRUE` (see details).

`eventTime` The assumed time under which the event rates are calculated, default is `12`.

`accrualTime` The assumed accrual time intervals for the study, default is `c(0,12)` (see details).

`accrualIntensity`
A vector of accrual intensities, default is the relative intensity `0.1` (see details).

`followUpTime` The assumed (additional) follow-up time for the study, default is `6`. The total study duration is `accrualTime + followUpTime`.

`maxNumberOfSubjects`
If `maxNumberOfSubjects > 0` is specified, the follow-up time for the required number of events is determined.

`dropoutRate1` The assumed drop-out rate in the treatment group, default is `0`.

`dropoutRate2` The assumed drop-out rate in the control 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 subjects for testing the hazard ratio in a survival design. It also calculates the time when the required events are expected under the given assumptions (exponentially, piecewise exponentially, or Weibull distributed survival times and constant or non-constant piecewise accrual). Additionally, an allocation ratio = n1/n2 can be specified where n1 and n2 are the number of subjects in the two treatment groups.

The formula of Kim & Tsiatis (Biometrics, 1990) is used to calculate the expected number of events under the alternative (see also Lakatos & Lan, Statistics in Medicine, 1992). These formulas are generalized to piecewise survival times and non-constant piecewise accrual over time.
If `accountForObservationTimes = FALSE`, only the event rates are used for the calculation of the maximum number of subjects.

`piecewiseSurvivalTime` The first element of this vector must be equal to `0`. `piecewiseSurvivalTime` can also be a list that combines the definition of the time intervals and hazard rates in the reference group. The definition of the survival time in the treatment group is obtained by the specification of the hazard ratio (see examples for details).

`accrualTime` can also be used to define a non-constant accrual over time. For this, `accrualTime` needs to be a vector that defines the accrual intervals and `accrualIntensity` needs to be specified. The first element of `accrualTime` must be equal to 0.
`accrualTime` can also be a list that combines the definition of the accrual time and accrual intensity `accrualIntensity` (see below and examples for details). If the length of `accrualTime` and the length of `accrualIntensity` are the same (i.e., the end of accrual is undefined), `maxNumberOfPatients > 0` needs to be specified and the end of accrual is calculated.

`accrualIntensity` needs to be defined if a vector of `accrualTime` is specified.
If the length of `accrualTime` and the length of `accrualIntensity` are the same (i.e., the end of accrual is undefined), `maxNumberOfPatients > 0` needs to be specified and the end of accrual is calculated. In that case, `accrualIntensity` is given by the number of subjects per time unit.
If the length of `accrualTime` equals the length of `accrualIntensity - 1` (i.e., the end of accrual is defined), `maxNumberOfPatients` is calculated. In that case, `accrualIntensity` defines the intensity how subjects enter the trial. For example, `accrualIntensity = c(1,2)` specifies that in the second accrual interval the intensity is doubled as compared to the first accrual interval. The actual accrual intensity is calculated for the calculated `maxNumberOfPatients`.

`accountForObservationTime` can be selected as `FALSE`. In this case, the number of subjects is calculated from the event probabilities only. This kind of computation does not account for the specific accrual pattern and survival distribution.

Value

Returns a `TrialDesignPlanSurvival` object.

Examples

```
# Fixed sample size trial with median survival 20 vs. 30 months in treatment and
# reference group, respectively, alpha = 0.05 (two-sided), and power 1 - beta = 90%.
# 20 subjects will be recruited per month up to 400 subjects, i.e., accrual time is 20 mo
getSampleSizeSurvival(alpha = 0.05, sided = 2, beta = 0.1, lambda1 = log(2) / 20,
    lambda2 = log(2) / 30, accrualTime = c(0,20), accrualIntensity = 20)
```

```
# Fixed sample size with minimum required definitions, pi1 = c(0.4,0.5,0.6) and
# pi2 = 0.2 at event time 12, accrual time 12 and follow-up time 6 as default,
# only alpha = 0.01 is specified
getSampleSizeSurvival(alpha = 0.01)

# Four stage O'Brien & Fleming group sequential design with minimum required
# definitions, pi1 = c(0.4,0.5,0.6) and pi2 = 0.2 at event time 12,
# accrual time 12 and follow-up time 6 as default
getSampleSizeSurvival(design = getDesignGroupSequential(kMax = 4))

# For fixed sample design, determine necessary accrual time if 200 subjects and
# 30 subjects per time unit can be recruited
getSampleSizeSurvival(accrualTime = c(0), accrualIntensity = c(30),
   maxNumberOfSubjects = 200)

# Determine necessary accrual time if 200 subjects and if the first 6 time units
# 20 subjects per time unit can be recruited, then 30 subjects per time unit
getSampleSizeSurvival(accrualTime = c(0, 6), accrualIntensity = c(20, 30),
   maxNumberOfSubjects = 200)

# Determine maximum number of Subjects if the first 6 time units 20 subjects
# per time unit can be recruited, and after 10 time units 30 subjects per time unit
getSampleSizeSurvival(accrualTime = c(0, 6, 10), accrualIntensity = c(20, 30))

# Specify accrual time as a list
at <- list(
    "0 - <6"  = 20,
    "6 - Inf" = 30)
getSampleSizeSurvival(accrualTime = at, maxNumberOfSubjects = 200)

# Specify accrual time as a list, if maximum number of subjects need to be calculated
at <- list(
    "0 - <6"   = 20,
    "6 - <=10" = 30)
getSampleSizeSurvival(accrualTime = at)

# Specify effect size for a two-stage group design with O'Brien & Fleming boundaries
# Effect size is based on event rates at specified event time
# needs to be specified because it should be shown that hazard ratio < 1
getSampleSizeSurvival(design = getDesignGroupSequential(kMax = 2),
   pi1 = 0.2, pi2 = 0.3, eventTime = 24)

# Effect size is based on event rate at specified event
# time for the reference group and hazard ratio
getSampleSizeSurvival(design = getDesignGroupSequential(kMax = 2),
   hazardRatio = 0.5, pi2 = 0.3, eventTime = 24)

# Effect size is based on hazard rate for the reference group and hazard ratio
getSampleSizeSurvival(design = getDesignGroupSequential(kMax = 2),
   hazardRatio = 0.5, lambda2 = 0.02)

# Specification of piecewise exponential survival time and hazard ratios
getSampleSizeSurvival(design = getDesignGroupSequential(kMax = 2),
   piecewiseSurvivalTime = c(0, 5, 10), lambda2 = c(0.01, 0.02, 0.04),
   hazardRatio = c(1.5, 1.8, 2))
```

```
# Specification of piecewise exponential survival time as a list and hazard ratios
pws <- list(
    "0 - <5"  = 0.01,
    "5 - <10" = 0.02,
    ">=10"    = 0.04)
getSampleSizeSurvival(design = getDesignGroupSequential(kMax = 2),
   piecewiseSurvivalTime = pws, hazardRatio = c(1.5, 1.8, 2))

# Specification of piecewise exponential survival time for both treatment arms
getSampleSizeSurvival(design = getDesignGroupSequential(kMax = 2),
   piecewiseSurvivalTime = c(0, 5, 10), lambda2 = c(0.01, 0.02, 0.04),
   lambda1 = c(0.015, 0.03, 0.06))

# Specification of piecewise exponential survival time as a list
pws <- list(
    "0 - <5"  = 0.01,
    "5 - <10" = 0.02,
    ">=10"    = 0.04)
getSampleSizeSurvival(design = getDesignGroupSequential(kMax = 2),
   piecewiseSurvivalTime = pws, hazardRatio = c(1.5, 1.8, 2))

# Specify effect size based on median survival times
median1 <- 5
median2 <- 3
getSampleSizeSurvival(lambda1 = log(2) / median1, lambda2 = log(2) / median2)

# Specify effect size based on median survival times of Weibull distribtion with kappa =
median1 <- 5
median2 <- 3
kappa <- 2
getSampleSizeSurvival(lambda1 = log(2)^(1 / kappa) / median1,
   lambda2 = log(2)^(1 / kappa) / median2, kappa = kappa)

# Identify minimal and maximal required subjects to
# reach the required events in spite of dropouts
getSampleSizeSurvival(accrualTime = c(0, 18), accrualIntensity = c(20, 30),
    lambda2 = 0.4, lambda1 = 0.3, followUpTime = Inf, dropoutRate1 = 0.001,
    dropoutRate2 = 0.005)
getSampleSizeSurvival(accrualTime = c(0, 18), accrualIntensity = c(20, 30),
    lambda2 = 0.4, lambda1 = 0.3, followUpTime = 0, dropoutRate1 = 0.001,
    dropoutRate2 = 0.005)
```

`getSimulationMeans` *Get Simulation Means*

Description

Returns the simulated power, stopping probabilities, conditional power, and expected sample size for testing means in a one or two treatment groups testing situation.

Usage

```
getSimulationMeans(design = NULL, ..., groups = 2L,
  meanRatio = FALSE, thetaH0 = ifelse(meanRatio, 1, 0),
  alternative = C_ALTERNATIVE_POWER_SIMULATION_DEFAULT,
  stDev = C_STDEV_DEFAULT, plannedSubjects = NA_real_,
  directionUpper = C_DIRECTION_UPPER_DEFAULT,
  allocationRatioPlanned = NA_real_,
  minNumberOfAdditionalSubjectsPerStage = NA_real_,
  maxNumberOfAdditionalSubjectsPerStage = NA_real_,
  conditionalPower = NA_real_, thetaH1 = NA_real_,
  maxNumberOfIterations = C_MAX_SIMULATION_ITERATIONS_DEFAULT,
  seed = NA_real_, calcSubjectsFunction = NULL)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, `alpha`, `beta`, and `sided` can be directly entered as argument.
`...`	Ensures that all arguments are be named and that a warning will be displayed if unknown arguments are passed.
`groups`	The number of treatment groups (1 or 2), default is `2`.
`meanRatio`	If `meanRatio = TRUE` is specified, the design characteristics for one-sided testing of H0: mu1/mu2 = thetaH0 are simulated, 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` or `1` for difference and ratio testing, respectively.
`alternative`	The alternative hypothesis value. This can be a vector of assumed alternatives, default is `seq(0,1,0.2)`.
`stDev`	The standard deviation under which the conditional power calculation is performed, default is 1. If `meanRatio = TRUE` is specified, stDev defines the coefficient of variation sigma/mu2.
`plannedSubjects`	`plannedSubjects` is a vector of length `kMax` (the number of stages of the design) that determines the number of cumulated (overall) subjects when the interim stages are planned.
`directionUpper`	Specifies the direction of the alternative, only applicable for one-sided testing, default is `TRUE`.
`allocationRatioPlanned`	The planned allocation ratio for a two treatment groups design, default is `1`.
`minNumberOfAdditionalSubjectsPerStage`	When performing a data driven sample size recalculation, the vector with length kMax `minNumberOfAdditionalSubjectsPerStage` determines the minimum number of subjects per stage (i.e., not cumulated), the first element is not taken into account.
`maxNumberOfAdditionalSubjectsPerStage`	When performing a data driven sample size recalculation, the vector with length kMax `maxNumberOfAdditionalSubjectsPerStage` determines the maximum number of subjects per stage (i.e., not cumulated), the first element is not taken into account.

conditionalPower
: The conditional power under which the sample size recalculation is performed.

thetaH1
: If specified, the value of the alternative under which the conditional power calculation is performed.

maxNumberOfIterations
: The number of simulation iterations.

seed
: The seed to reproduce the simulation, default is a random seed.

calcSubjectsFunction
: Optionally, a function can be entered that defines the way of performing the sample size recalculation. By default, sample size recalulation is performed with conditional power with specified minNumberOfAdditionalSubjectsPerStage and maxNumberOfAdditionalSubjectsPerStage (see details and examples).

Details

At given design the function simulates the power, stopping probabilities, conditional power, and expected sample size at given number of subjects and parameter configuration. Additionally, an allocation ratio = n1/n2 can be specified where n1 and n2 are the number of subjects in the two treatment groups.

calcSubjectsFunction
This function returns the number of subjects at given conditional power and conditional Type I error rate for specified testing situation. The function might depend on variables stage, meanRatio, thetaH0, groups, plannedSubjects, sampleSizesPerStage, directionUpper, allocationRatioPlanned, minNumberOfAdditionalSubjectsPerStage, maxNumberOfAdditiona conditionalPower, conditionalCriticalValue, thetaStandardized. The function has to obtain the three-dots arument '...' (see examples).

Value

Returns a SimulationResultsMeans object.

Simulation Data

The summary statistics "Simulated data" contains the following parameters: median [range]; mean +/-sd

$show(showStatistics = FALSE) or $setShowStatistics(FALSE) can be used to disable the output of the aggregated simulated data.

Example 1:

```
simulationResults <- getSimulationMeans(plannedSubjects = 40)
simulationResults$show(showStatistics = FALSE)
```

Example 2:

```
simulationResults <- getSimulationMeans(plannedSubjects = 40)
simulationResults$setShowStatistics(FALSE)
simulationResults
```

getData can be used to get the aggregated simulated data from the object as data.frame. The data frame contains the following columns:

1. `iterationNumber`: The number of the simulation iteration.
2. `stageNumber`: The stage.
3. `alternative`: The alternative hypothesis value.
4. `numberOfSubjects`: The number of subjects under consideration when the (interim) analysis takes place.
5. `rejectPerStage`: 1 if null hypothesis can be rejected, 0 otherwise.
6. `futilityPerStage`: 1 if study should be stopped for futility, 0 otherwise.
7. `testStatistic`: The test statistic that is used for the test decision, depends on which design was chosen (group sequential, inverse normal, or Fishers combination test).
8. `testStatisticsPerStage`: The test statistic for each stage if only data from the considered stage is taken into account.
9. `effectEstimate`: Standardized overall simulated effect estimate.
10. `trialStop`: `TRUE` if study should be stopped for efficacy or futility or final stage, `FALSE` otherwise.
11. `conditionalPowerAchieved`: The conditional power for the subsequent stage of the trial for selected sample size and effect. The effect is either estimated from the data or can be user defined with `thetaH1`.

Examples

```
# Fixed sample size with minimum required definitions,
# alternative = c(0, 1, 2, 3, 4), standard deviation = 5
getSimulationMeans(getDesignGroupSequential(), alternative = 40,
    stDev = 50, plannedSubjects = c(20, 40, 60), thetaH1 = 60,
    maxNumberOfIterations = 50)



# Increase number of simulation iterations and compare results
# with power calculator using normal approximation
getSimulationMeans(alternative = 0:4, stDev = 5,
    plannedSubjects = 40, maxNumberOfIterations = 50)
getPowerMeans(alternative = 0:4, stDev = 5,
    maxNumberOfSubjects = 40, normalApproximation = TRUE)

# Do the same for a three-stage O'Brien&Fleming inverse
# normal group sequential design with non-binding futility stops
designIN <- getDesignInverseNormal(typeOfDesign = "OF", futilityBounds = c(0, 0))
x <- getSimulationMeans(designIN, alternative = c(0:4), stDev = 5,
    plannedSubjects = c(20, 40, 60), maxNumberOfIterations = 1000)
getPowerMeans(designIN, alternative = 0:4, stDev = 5,
    maxNumberOfSubjects = 60, normalApproximation = TRUE)

# Assess power and average sample size if a sample size increase is foreseen
# at conditional power 80% for each subsequent stage based on observed overall
# effect and specified minNumberOfAdditionalSubjectsPerStage and
# maxNumberOfAdditionalSubjectsPerStage
getSimulationMeans(designIN, alternative = 0:4, stDev = 5,
    plannedSubjects = c(20, 40, 60),
    minNumberOfAdditionalSubjectsPerStage = c(20, 20, 20),
    maxNumberOfAdditionalSubjectsPerStage = c(80, 80, 80),
```

```
    conditionalPower = 0.8,maxNumberOfIterations = 50)

# Do the same under the assumption that a sample size increase only takes
# place at the first interim. The sample size for the third stage is set equal
# to the second stage sample size.
mySampleSizeCalculationFunction <- function(..., stage,
        minNumberOfAdditionalSubjectsPerStage,
        maxNumberOfAdditionalSubjectsPerStage,
        sampleSizesPerStage,
        conditionalPower,
        conditionalCriticalValue,
        thetaStandardized) {
    if (stage == 2) {
        stageSubjects <- 4 * (max(0, conditionalCriticalValue +
            stats::qnorm(conditionalPower)))^2 / (max(1e-12, thetaStandardized))^2
        stageSubjects <- min(max(minNumberOfAdditionalSubjectsPerStage[stage],
            stageSubjects), maxNumberOfAdditionalSubjectsPerStage[stage])
    } else {
        stageSubjects <- sampleSizesPerStage[stage - 1]
    }
    return(stageSubjects)
}
getSimulationMeans(designIN, alternative = 2:4, stDev = 5,
    plannedSubjects = c(20, 40, 60),
    minNumberOfAdditionalSubjectsPerStage = c(20, 20, 2),
    maxNumberOfAdditionalSubjectsPerStage = c(40, 160, 16),
    conditionalPower = 0.8,
    calcSubjectsFunction = mySampleSizeCalculationFunction,
    maxNumberOfIterations = 50)
```

`getSimulationRates` *Get Simulation Rates*

Description

Returns the simulated power, stopping probabilities, conditional power, and expected sample size for testing rates in a one or two treatment groups testing situation.

Usage

```
getSimulationRates(design = NULL, ..., groups = 2L,
  riskRatio = FALSE, thetaH0 = ifelse(riskRatio, 1, 0),
  pi1 = C_PI_1_DEFAULT, pi2 = NA_real_, plannedSubjects = NA_real_,
  directionUpper = C_DIRECTION_UPPER_DEFAULT,
  allocationRatioPlanned = NA_real_,
  minNumberOfAdditionalSubjectsPerStage = NA_real_,
  maxNumberOfAdditionalSubjectsPerStage = NA_real_,
  conditionalPower = NA_real_, pi1H1 = NA_real_, pi2H1 = 0.2,
  maxNumberOfIterations = C_MAX_SIMULATION_ITERATIONS_DEFAULT,
  seed = NA_real_, calcSubjectsFunction = NULL)
```

Arguments

design	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, alpha, beta, and sided can be directly entered as argument.
...	Ensures that all arguments are be named and that a warning will be displayed if unknown arguments are passed.
groups	The number of treatment groups (1 or 2), default is 2.
riskRatio	If riskRatio = TRUE is specified, the design characteristics for one-sided testing of H0: pi1/pi2 = thetaH0 are simulated, 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 or 1 for difference and ratio testing, respectively.
pi1	The assumed probability in the active treatment group if two treatment groups are considered, or the alternative probability for a one treatment group design, default is seq(0.2,0.5,0.1).
pi2	The assumed probability in the reference group if two treatment groups are considered, default is 0.2.
plannedSubjects	plannedSubjects is a vector of length kMax (the number of stages of the design) that determines the number of cumulated (overall) subjects when the interim stages are planned.
directionUpper	Specifies the direction of the alternative, only applicable for one-sided testing, default is TRUE.
allocationRatioPlanned	The planned allocation ratio for a two treatment groups design, default is 1.
minNumberOfAdditionalSubjectsPerStage	When performing a data driven sample size recalculation, the vector with length kMax minNumberOfAdditionalSubjectsPerStage determines the minimum number of subjects per stage (i.e., not cumulated), the first element is not taken into account.
maxNumberOfAdditionalSubjectsPerStage	When performing a data driven sample size recalculation, the vector with length kMax maxNumberOfAdditionalSubjectsPerStage determines the maximum number of subjects per stage (i.e., not cumulated), the first element is not taken into account.
conditionalPower	The conditional power under which the sample size recalculation is performed.
pi1H1	If specified, the assumed probability in the active treatment group if two treatment groups are considered, or the assumed probability for a one treatment group design, for which the conditional power was calculated.
pi2H1	If specified, the assumed probability in the reference group if two treatment groups are considered, for which the conditional power was calculated, default is 0.2.
maxNumberOfIterations	The number of simulation iterations.
seed	The seed to reproduce the simulation, default is a random seed.
calcSubjectsFunction	Optionally, a function can be entered that defines the way of performing the sample size recalculation. By default, sample size recalulation is performed with

conditional power and specified `minNumberOfAdditionalSubjectsPerStage` and `maxNumberOfAdditionalSubjectsPerStage` (see details and examples).

Details

At given design the function simulates the power, stopping probabilities, conditional power, and expected sample size at given number of subjects and parameter configuration. Additionally, an allocation ratio = n1/n2 can be specified where n1 and n2 are the number of subjects in the two treatment groups.

calcSubjectsFunction
This function returns the number of subjects at given conditional power and conditional Type I error rate for specified testing situation. The function might depend on variables `stage`, `riskRatio`, `thetaH0`, `groups`, `plannedSubjects`, `directionUpper`, `allocationRatioPlanned`, `minNumberOfAdditionalSubjectsPerStage`, `maxNumberOfAdditionalSubjectsPerStage`, `sampleSizesPerStage`, `conditionalPower`, `conditionalCriticalValue`, `overallRate`, `farringtonManningValue1`, and `farringtonManningValue2`. The function has to obtain the three-dots arument '...' (see examples).

Value

Returns a `SimulationResultsRates` object.

Simulation Data

The summary statistics "Simulated data" contains the following parameters: median [range]; mean +/-sd

`$show(showStatistics = FALSE)` or `$setShowStatistics(FALSE)` can be used to disable the output of the aggregated simulated data.

Example 1:

```
simulationResults <- getSimulationRates(plannedSubjects = 40)
simulationResults$show(showStatistics = FALSE)
```

Example 2:

```
simulationResults <- getSimulationRates(plannedSubjects = 40)
simulationResults$setShowStatistics(FALSE)
simulationResults
```

`getData` can be used to get the aggregated simulated data from the object as `data.frame`. The data frame contains the following columns:

1. `iterationNumber`: The number of the simulation iteration.
2. `stageNumber`: The stage.
3. `pi1`: The assumed or derived event rate in the treatment group (if available).
4. `pi2`: The assumed or derived event rate in the control group (if available).
5. `numberOfSubjects`: The number of subjects under consideration when the (interim) analysis takes place.
6. `rejectPerStage`: 1 if null hypothesis can be rejected, 0 otherwise.

7. `futilityPerStage`: 1 if study should be stopped for futility, 0 otherwise.
8. `testStatistic`: The test statistic that is used for the test decision, depends on which design was chosen (group sequential, inverse normal, or Fisher combination test)'
9. `testStatisticsPerStage`: The test statistic for each stage if only data from the considered stage is taken into account.
10. `overallRates1`: The overall rate in treatment group 1.
11. `overallRates2`: The overall rate in treatment group 2.
12. `stagewiseRates1`: The stagewise rate in treatment group 1.
13. `stagewiseRates2`: The stagewise rate in treatment group 2.
14. `sampleSizesPerStage1`: The stagewise sample size in treatment group 1.
15. `sampleSizesPerStage2`: The stagewise sample size in treatment group 2.
16. `trialStop`: `TRUE` if study should be stopped for efficacy or futility or final stage, `FALSE` otherwise.
17. `conditionalPowerAchieved`: The conditional power for the subsequent stage of the trial for selected sample size and effect. The effect is either estimated from the data or can be user defined with `pi1H1` and `pi2H1`.

Examples

```
# Fixed sample size with minimum required definitions, pi1 = (0.3,0.4,0.5, 0.6) and pi2 =
getSimulationRates(pi1 = seq(0.3, 0.6, 0.1), pi2 = 0.3,
    plannedSubjects = 120, maxNumberOfIterations = 50)



# Increase number of simulation iterations and compare results with power calculator
getSimulationRates(pi1 = seq(0.3, 0.6, 0.1), pi2 = 0.3,
    plannedSubjects = 120, maxNumberOfIterations = 50)
getPowerRates(pi1 = seq(0.3, 0.6, 0.1), pi2 = 0.3, maxNumberOfSubjects = 120)

# Do the same for a two-stage Pocock inverse normal group sequential
# design with non-binding futility stops
designIN <- getDesignInverseNormal(typeOfDesign = "P", futilityBounds = c(0))
getSimulationRates(designIN, pi1 = seq(0.3, 0.6, 0.1), pi2 = 0.3,
    plannedSubjects = c(40, 80), maxNumberOfIterations = 50)
getPowerRates(designIN, pi1 = seq(0.3, 0.6, 0.1), pi2 = 0.3, maxNumberOfSubjects = 80)

# Assess power and average sample size if a sample size reassessment is
# foreseen at conditional power 80% for the subsequent stage (decrease and increase)
# based on observed overall rates and specified minNumberOfAdditionalSubjectsPerStage
# and maxNumberOfAdditionalSubjectsPerStage

# Do the same under the assumption that a sample size increase only takes place
# if the rate difference exceeds the value 0.1 at interim. For this, the sample
# size recalculation method needs to be redefined:
mySampleSizeCalculationFunction <- function(..., stage,
        plannedSubjects,
        minNumberOfAdditionalSubjectsPerStage,
        maxNumberOfAdditionalSubjectsPerStage,
        conditionalPower,
        conditionalCriticalValue,
```

```
        overallRate) {
    if (overallRate[1] - overallRate[2] < 0.1) {
        return(plannedSubjects[stage] - plannedSubjects[stage - 1])
    } else {
        rateUnderH0 <- (overallRate[1] + overallRate[2]) / 2
        stageSubjects <- 2 * (max(0, conditionalCriticalValue *
            sqrt(2 * rateUnderH0 * (1 - rateUnderH0)) +
            stats::qnorm(conditionalPower) * sqrt(overallRate[1] *
            (1 - overallRate[1]) + overallRate[2] * (1 - overallRate[2])))^2 /
            (max(1e-12,(overallRate[1] - overallRate[2])))^2
        stageSubjects <- ceiling(min(max(
            minNumberOfAdditionalSubjectsPerStage[stage],
            stageSubjects), maxNumberOfAdditionalSubjectsPerStage[stage]))
        return(stageSubjects)
    }
}
getSimulationRates(designIN, pi1 = seq(0.3, 0.6, 0.1), pi2 = 0.3,
    plannedSubjects = c(40, 80), minNumberOfAdditionalSubjectsPerStage = c(40, 20),
    maxNumberOfAdditionalSubjectsPerStage = c(40, 160), conditionalPower = 0.8,
    calcSubjectsFunction = mySampleSizeCalculationFunction, maxNumberOfIterations = 50)
```

`getSimulationSurvival`

Get Simulation Survival

Description

Returns the analysis times, power, stopping probabilities, conditional power, and expected sample size for testing the hazard ratio in a two treatment groups survival design.

Usage

```
getSimulationSurvival(design = NULL, ...,
  thetaH0 = C_THETA_H0_SURVIVAL_DEFAULT,
  directionUpper = C_DIRECTION_UPPER_DEFAULT, pi1 = NA_real_,
  pi2 = NA_real_, lambda1 = NA_real_, lambda2 = NA_real_,
  hazardRatio = NA_real_, kappa = 1,
  piecewiseSurvivalTime = NA_real_,
  allocation1 = C_ALLOCATION_1_DEFAULT,
  allocation2 = C_ALLOCATION_2_DEFAULT,
  eventTime = C_EVENT_TIME_DEFAULT,
  accrualTime = C_ACCRUAL_TIME_DEFAULT,
  accrualIntensity = C_ACCRUAL_INTENSITY_DEFAULT,
  dropoutRate1 = C_DROP_OUT_RATE_1_DEFAULT,
  dropoutRate2 = C_DROP_OUT_RATE_2_DEFAULT,
  dropoutTime = C_DROP_OUT_TIME_DEFAULT,
  maxNumberOfSubjects = NA_real_, plannedEvents = NA_real_,
  minNumberOfAdditionalEventsPerStage = NA_real_,
  maxNumberOfAdditionalEventsPerStage = NA_real_,
  conditionalPower = NA_real_, thetaH1 = NA_real_,
```

```
maxNumberOfIterations = C_MAX_SIMULATION_ITERATIONS_DEFAULT,
maxNumberOfRawDatasetsPerStage = 0,
longTimeSimulationAllowed = FALSE, seed = NA_real_)
```

Arguments

design	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, alpha, beta, twoSidedPower, and sided can be directly entered as argument.
...	Ensures that all arguments are be named and that a warning will be displayed if unknown arguments are passed.
thetaH0	The null hypothesis value. The default value is 1. For one-sided testing, a bound for testing H0: hazard ratio = thetaH0 != 1 can be specified.
directionUpper	Specifies the direction of the alternative, only applicable for one-sided testing, default is TRUE.
pi1	The assumed event rate in the treatment group, default is seq(0.2,0.5,0.1).
pi2	The assumed event rate in the control group, default is 0.2.
lambda1	The assumed hazard rate in the treatment group, there is no default. lambda1 can also be used to define piecewise exponentially distributed survival times (see details).
lambda2	The assumed hazard rate in the reference group, there is no default. lambda2 can also be used to define piecewise exponentially distributed survival times (see details).
hazardRatio	The vector of hazard ratios under consideration. If the event or hazard rates in both treatment groups are defined, the hazard ratio needs not to be specified as it is calculated.
kappa	The scale parameter of the Weibull distribution, default is 1. The Weibull distribution cannot be used for the piecewise definition of the survival time distribution. Note that the parameters shape and scale in Weibull are equivalent to kappa and 1 / lambda, respectively, in rpact.
piecewiseSurvivalTime	A vector that specifies the time intervals for the piecewise definition of the exponential survival time cumulative distribution function (see details).
allocation1	The number how many subjects are assigned to treatment 1 in a subsequent order, default is 1
allocation2	The number how many subjects are assigned to treatment 2 in a subsequent order, default is 1
eventTime	The assumed time under which the event rates are calculated, default is 12.
accrualTime	The assumed accrual time for the study, default is 12 (see getAccrualTime).
accrualIntensity	A vector of accrual intensities, default is the relative intensity 0.1 (see getAccrualTime).
dropoutRate1	The assumed drop-out rate in the treatment group, default is 0.
dropoutRate2	The assumed drop-out rate in the control group, default is 0.
dropoutTime	The assumed time for drop-out rates in the control and the treatment group, default is 12.

`maxNumberOfSubjects`
: `maxNumberOfSubjects > 0` needs to be specified. If accrual time and accrual intensity is specified, this will be calculated.

`plannedEvents`
: `plannedEvents` is a vector of length kMax (the number of stages of the design) with increasing numbers that determines the number of cumulated (overall) events when the interim stages are planned.

`minNumberOfAdditionalEventsPerStage`
: When performing a data driven sample size recalculation, the vector with length kMax `minNumberOfAdditionalEventsPerStage` determines the minimum number of events per stage (i.e., not cumulated), the first element is not taken into account.

`maxNumberOfAdditionalEventsPerStage`
: When performing a data driven sample size recalculation, the vector with length kMax `maxNumberOfAdditionalEventsPerStage` determines the maximum number of events per stage (i.e., not cumulated), the first element is not taken into account.

`conditionalPower`
: The conditional power under which the sample size recalculation is performed.

`thetaH1`
: If specified, the value of the hazard ratio under which the conditional power calculation is performed.

`maxNumberOfIterations`
: The number of simulation iterations.

`maxNumberOfRawDatasetsPerStage`
: The number of raw datasets per stage that shall be extracted and saved as `data.frame`, default is `0`. `getRawData` can be used to get the extracted raw data from the object.

`longTimeSimulationAllowed`
: Logical that indicates whether long time simulations that consumes more than 30 seconds are allowed or not, default is `FALSE`.

`seed`
: The seed to reproduce the simulation, default is a random seed.

Details

At given design the function simulates the power, stopping probabilities, conditional power, and expected sample size at given number of events, number of subjects, and parameter configuration. It also simulates the time when the required events are expected under the given assumptions (exponentially, piecewise exponentially, or Weibull distributed survival times and constant or non-constant piecewise accrual). Additionally, integers `allocation1` and `allocation2` can be specified that determine the number allocated to treatment group 1 and treatment group 2, respectively.

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). These formulas are generalized to piecewise survival times and non-constant piecewise accrual over time.

`piecewiseSurvivalTime` The first element of this vector must be equal to `0`. `piecewiseSurvivalTime` can also be a list that combines the definition of the time intervals and hazard rates in the reference group. The definition of the survival time in the treatment group is obtained by the specification of the hazard ratio (see examples for details).

Value

Returns a SimulationResultsSurvival object.

Simulation Data

The summary statistics "Simulated data" contains the following parameters: median [range]; mean +/-sd

$show(showStatistics = FALSE) or $setShowStatistics(FALSE) can be used to disable the output of the aggregated simulated data.

Example 1:

```
simulationResults <- getSimulationSurvival(maxNumberOfSubjects = 100, plannedEve
simulationResults$show(showStatistics = FALSE)
```

Example 2:

```
simulationResults <- getSimulationSurvival(maxNumberOfSubjects = 100, plannedEve
simulationResults$setShowStatistics(FALSE)
simulationResults
```

getData can be used to get the aggregated simulated data from the object as data.frame. The data frame contains the following columns:

1. iterationNumber: The number of the simulation iteration.
2. stageNumber: The stage.
3. pi1: The assumed or derived event rate in the treatment group.
4. pi2: The assumed or derived event rate in the control group.
5. hazardRatio: The hazard ratio under consideration (if available).
6. analysisTime: The analysis time.
7. numberOfSubjects: The number of subjects under consideration when the (interim) analysis takes place.
8. eventsPerStage1: The observed number of events per stage in treatment group 1.
9. eventsPerStage2: The observed number of events per stage in treatment group 2.
10. eventsPerStage: The observed number of events per stage in both treatment groups.
11. rejectPerStage: 1 if null hypothesis can be rejected, 0 otherwise.
12. futilityPerStage: 1 if study should be stopped for futility, 0 otherwise.
13. eventsNotAchieved: 1 if number of events could not be reached with observed number of subjects, 0 otherwise.
14. testStatistic: The test statistic that is used for the test decision, depends on which design was chosen (group sequential, inverse normal, or Fisher combination test)'
15. logRankStatistic: Z-score statistic which corresponds to a one-sided log-rank test at considered stage.
16. hazardRatioEstimateLR: The estimated hazard ratio, derived from the log-rank statistic.
17. trialStop: TRUE if study should be stopped for efficacy or futility or final stage, FALSE otherwise.
18. conditionalPowerAchieved: The conditional power for the subsequent stage of the trial for selected sample size and effect. The effect is either estimated from the data or can be user defined with thetaH1.

Raw Data

`getRawData` can be used to get the simulated raw data from the object as `data.frame`. Note that `getSimulationSurvival` must called before with `maxNumberOfRawDatasetsPerStage` > 0. The data frame contains the following columns:

1. `iterationNumber`: The number of the simulation iteration.
2. `stopStage`: The stage of stopping.
3. `subjectId`: The subject id (increasing number 1, 2, 3, ...)
4. `accrualTime`: The accrual time, i.e., the time when the subject entered the trial.
5. `treatmentGroup`: The treatment group number (1 or 2).
6. `survivalTime`: The survival time of the subject.
7. `dropoutTime`: The dropout time of the subject (may be `NA`).
8. `observationTime`: The specific observation time.
9. `timeUnderObservation`: The time under observation is defined as follows:
   if (event == TRUE)
   timeUnderObservation <- survivalTime;
   else if (dropoutEvent == TRUE)
   timeUnderObservation <- dropoutTime;
   else
   timeUnderObservation <- observationTime - accrualTime;
10. `event`: `TRUE` if an event occurred; `FALSE` otherwise.
11. `dropoutEvent`: `TRUE` if an dropout event occurred; `FALSE` otherwise.

Examples

```
# Fixed sample size with minimum required definitions, pi1 = (0.3,0.4,0.5,0.6) and
# pi2 = 0.3 at event time 12, and accrual time 24
getSimulationSurvival(pi1 = seq(0.3,0.6,0.1), pi2 = 0.3, eventTime = 12,
    accrualTime = 24, plannedEvents = 40, maxNumberOfSubjects = 200,
    maxNumberOfIterations = 50)



# Increase number of simulation iterations
getSimulationSurvival(pi1 = seq(0.3,0.6,0.1), pi2 = 0.3, eventTime = 12,
    accrualTime = 24, plannedEvents = 40, maxNumberOfSubjects = 200,
    maxNumberOfIterations = 50)

# Determine necessary accrual time with default settings if 200 subjects and
# 30 subjects per time unit can be recruited
getSimulationSurvival(plannedEvents = 40, accrualTime = 0,
    accrualIntensity = 30, maxNumberOfSubjects = 200, maxNumberOfIterations = 50)

# Determine necessary accrual time with default settings if 200 subjects and
# if the first 6 time units 20 subjects per time unit can be recruited,
# then 30 subjects per time unit
getSimulationSurvival(plannedEvents = 40, accrualTime = c(0, 6),
    accrualIntensity = c(20, 30), maxNumberOfSubjects = 200,
    maxNumberOfIterations = 50)
```

```
# Determine maximum number of Subjects with default settings if the first
# 6 time units 20 subjects per time unit can be recruited, and after
# 10 time units 30 subjects per time unit
getSimulationSurvival(plannedEvents = 40, accrualTime = c(0, 6, 10),
    accrualIntensity = c(20, 30), maxNumberOfIterations = 50)

# Specify accrual time as a list
at <- list(
   "0 - <6"  = 20,
   "6 - Inf" = 30)
getSimulationSurvival(plannedEvents = 40, accrualTime = at,
    maxNumberOfSubjects = 200, maxNumberOfIterations = 50)

# Specify accrual time as a list, if maximum number of subjects need to be calculated
at <- list(
   "0 - <6"   = 20,
   "6 - <=10" = 30)
getSimulationSurvival(plannedEvents = 40, accrualTime = at, maxNumberOfIterations = 50)

# Specify effect size for a two-stage group sequential design with O'Brien & Fleming boun
# Effect size is based on event rates at specified event time, directionUpper = FALSE
# needs to be specified because it should be shown that hazard ratio < 1
getSimulationSurvival(design = getDesignGroupSequential(kMax = 2),
   pi1 = 0.2, pi2 = 0.3, eventTime = 24, plannedEvents = c(20, 40),
   maxNumberOfSubjects = 200, directionUpper = FALSE, maxNumberOfIterations = 50)

# As above, but with a three-stage O'Brien and Flemming design with
# specified information rates, note that planned events consists of integer values
d3 <- getDesignGroupSequential(informationRates = c(0.4, 0.7, 1))
getSimulationSurvival(design = d3, pi1 = 0.2, pi2 = 0.3, eventTime = 24,
   plannedEvents = round(d3$informationRates * 40),
   maxNumberOfSubjects = 200, directionUpper = FALSE,
    maxNumberOfIterations = 50)

# Effect size is based on event rate at specified event time for the reference group and
# hazard ratio, directionUpper = FALSE needs to be specified because it should be shown
# that hazard ratio < 1
getSimulationSurvival(design = getDesignGroupSequential(kMax = 2), hazardRatio = 0.5,
   pi2 = 0.3, eventTime = 24, plannedEvents = c(20, 40), maxNumberOfSubjects = 200,
   directionUpper = FALSE, maxNumberOfIterations = 50)

# Effect size is based on hazard rate for the reference group and
# hazard ratio, directionUpper = FALSE needs to be specified because
# it should be shown that hazard ratio < 1
getSimulationSurvival(design = getDesignGroupSequential(kMax = 2),
    hazardRatio = 0.5, lambda2 = 0.02, plannedEvents = c(20, 40),
    maxNumberOfSubjects = 200, directionUpper = FALSE,
    maxNumberOfIterations = 50)

# Specification of piecewise exponential survival time and hazard ratios,
# note that in getSimulationSurvival only on hazard ratio is used
# in the case that the survival time is piecewise expoential
getSimulationSurvival(design = getDesignGroupSequential(kMax = 2),
   piecewiseSurvivalTime = c(0, 5, 10), lambda2 = c(0.01, 0.02, 0.04),
   hazardRatio = 1.5, plannedEvents = c(20, 40), maxNumberOfSubjects = 200,
    maxNumberOfIterations = 50)
```

```
pws <- list(
    "0 - <5"  = 0.01,
   "5 - <10" = 0.02,
   ">=10"    = 0.04)
getSimulationSurvival(design = getDesignGroupSequential(kMax = 2),
   piecewiseSurvivalTime = pws, hazardRatio = c(1.5, 1.8, 2),
   plannedEvents = c(20, 40), maxNumberOfSubjects = 200,
    maxNumberOfIterations = 50)

# Specification of piecewise exponential survival time for both treatment arms
getSimulationSurvival(design = getDesignGroupSequential(kMax = 2),
   piecewiseSurvivalTime = c(0, 5, 10), lambda2 = c(0.01, 0.02, 0.04),
   lambda1 = c(0.015, 0.03, 0.06), plannedEvents = c(20, 40),
   maxNumberOfSubjects = 200, maxNumberOfIterations = 50)

# Specification of piecewise exponential survival time as a list,
# note that in getSimulationSurvival only on hazard ratio
# (not a vector) can be used
pws <- list(
    "0 - <5"  = 0.01,
   "5 - <10" = 0.02,
   ">=10"    = 0.04)
getSimulationSurvival(design = getDesignGroupSequential(kMax = 2),
   piecewiseSurvivalTime = pws, hazardRatio = 1.5,
   plannedEvents = c(20, 40), maxNumberOfSubjects = 200,
    maxNumberOfIterations = 50)

# Specification of piecewise exponential survival time and delayed effect
# (response after 5 time units)
getSimulationSurvival(design = getDesignGroupSequential(kMax = 2),
    piecewiseSurvivalTime = c(0, 5, 10), lambda2 = c(0.01, 0.02, 0.04),
    lambda1 = c(0.01, 0.02, 0.06), plannedEvents = c(20, 40),
    maxNumberOfSubjects = 200, maxNumberOfIterations = 50)

# Specify effect size based on median survival times
median1 <- 5
median2 <- 3
getSimulationSurvival(lambda1 = log(2) / median1,
   lambda2 = log(2) / median2, plannedEvents = 40,
   maxNumberOfSubjects = 200, directionUpper = FALSE,
    maxNumberOfIterations = 50)

# Specify effect size based on median survival
# times of Weibull distribtion with kappa = 2
median1 <- 5
median2 <- 3
kappa <- 2
getSimulationSurvival(lambda1 = log(2)^(1 / kappa) / median1,
   lambda2 = log(2)^(1 / kappa) / median2, kappa = kappa,
   plannedEvents = 40, maxNumberOfSubjects = 200,
    directionUpper = FALSE, maxNumberOfIterations = 50)

# Perform recalculation of number of events based on conditional power for a
# three-stage design with inverse normal combination test, where the conditional power
# is calculated under the specified effect size thetaH1 = 1.3 and up to a four-fold
# increase in originally planned sample size (number of events) is allowed
# Note that the first value in minNumberOfAdditionalEventsPerStage and
```

```
# maxNumberOfAdditionalEventsPerStage is arbitrary, i.e., it has no effect.
dIN <- getDesignInverseNormal(informationRates = c(0.4, 0.7, 1))

resultsWithSSR1 <- getSimulationSurvival(design = dIN,
    hazardRatio = seq(1, 1.6, 0.1),
    pi2 = 0.3, conditionalPower = 0.8, thetaH1 = 1.3,
    plannedEvents = c(58, 102, 146),
   minNumberOfAdditionalEventsPerStage = c(58, 44, 44),
    maxNumberOfAdditionalEventsPerStage = 4 * c(58, 44, 44),
   maxNumberOfSubjects = 800, maxNumberOfIterations = 50)
resultsWithSSR1

# If thetaH1 is unspecified, the observed hazard ratio estimate
# (calculated from the log-rank statistic) is used for performing the
# recalculation of the number of events
resultsWithSSR2 <- getSimulationSurvival(design = dIN,
    hazardRatio = seq(1, 1.6, 0.1),
   pi2 = 0.3, conditionalPower = 0.8, plannedEvents = c(58, 102, 146),
   minNumberOfAdditionalEventsPerStage = c(58, 44, 44),
    maxNumberOfAdditionalEventsPerStage = 4 * c(58, 44, 44),
   maxNumberOfSubjects = 800, maxNumberOfIterations = 50)
resultsWithSSR2

# Compare it with design without event size recalculation
resultsWithoutSSR <- getSimulationSurvival(design = dIN,
    hazardRatio = seq(1, 1.6, 0.1), pi2 = 0.3,
    plannedEvents = c(58, 102, 145), maxNumberOfSubjects = 800,
   maxNumberOfIterations = 50)
resultsWithoutSSR$overallReject
resultsWithSSR1$overallReject
resultsWithSSR2$overallReject

# Confirm that event size racalcuation increases the Type I error rate,
# i.e., you have to use the combination test
dGS <- getDesignGroupSequential(informationRates = c(0.4, 0.7, 1))
resultsWithSSRGS <- getSimulationSurvival(design = dGS, hazardRatio = seq(1),
   pi2 = 0.3, conditionalPower = 0.8, plannedEvents = c(58, 102, 145),
   minNumberOfAdditionalEventsPerStage = c(58, 44, 44),
    maxNumberOfAdditionalEventsPerStage = 4 * c(58, 44, 44),
   maxNumberOfSubjects = 800, maxNumberOfIterations = 50)
resultsWithSSRGS$overallReject

# Set seed to get reproduceable results

identical(
   getSimulationSurvival(plannedEvents = 40, maxNumberOfSubjects = 200,
        seed = 99)$analysisTime,
   getSimulationSurvival(plannedEvents = 40, maxNumberOfSubjects = 200,
        seed = 99)$analysisTime
)
```

`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 non-inferiority designs, this is the non-inferiority 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)
```

---

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.

NumberOfSubjects *Number Of Subjects*

Description

Class for definition of number of subjects results.

Details

NumberOfSubjects is a class for definition of number of subjects results.

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, ..., markdown = FALSE)
```

Arguments

x	The object to print.
markdown	If TRUE, the object x will be printed using markdown syntax; normal representation will be used otherwise (default is FALSE)

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.

PiecewiseSurvivalTime
Piecewise Exponential Survival Time

Description

Class for definition of piecewise survival times.

Details

`PiecewiseSurvivalTime` is a class for definition of piecewise survival times.

plot.AnalysisResults
Analysis Results Plotting

Description

Plots the conditional power together with the likelihood function.

Usage

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

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. Furthermore the following arguments can be defined: • thetaRange: A range of assumed effect sizes if testing means or a survival design was specified. Additionally, if testing means was selected, an assumed standard deviation can be specified (default is 1). • piRange: A range of assumed rates pi1 to calculate the conditional power. Additionally, if a two-sample comparison was selected, pi2 can be specified (default is the value from getAnalysisResults). • 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.
type	The plot type (default = 1). Note that at the moment only one type (the conditional power plot) is available.
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
showSource	If TRUE, the parameter names of the object will be printed which were used to create the plot; that may be, e.g., useful to check the values or to create own plots with plot.

Details

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

Examples

```
design <- getDesignGroupSequential(kMax = 2)

dataExample <- getDataset(
    n = c(20, 30),
   means = c(50, 51),
   stDevs = c(130, 140)
)

result <- getAnalysisResults(design = design,
   dataInput = dataExample, thetaH0 = 20,
   nPlanned = c(30), thetaH1 = 1.5, stage = 1)

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

---

plot.Dataset *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",
  showSource = FALSE)
```

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".
showSource	If TRUE, the parameter names of the object will be printed which were used to create the plot; that may be, e.g., useful to check the values or to create own plots with plot.

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")
```

---

plot.SimulationResults

Simulation Results Plotting

Description

Plots simulation results.

Usage

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

Arguments

x	The simulation results, obtained from getSimulationSurvival.
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 'Boundaries' plot
- `2`: creates a 'Boundaries Effect Scale' plot
- `3`: creates a 'Boundaries p Values Scale' plot
- `4`: creates a 'Type One Error Spending' plot
- `5`: creates a 'Sample Size' or 'Overall Power and Early Stopping' plot
- `6`: creates a 'Number of Events' or 'Sample Size' plot
- `7`: creates an 'Overall Power' plot
- `8`: creates an 'Overall Early Stopping' plot
- `9`: creates an 'Expected Number of Events' or 'Expected Sample Size' plot
- `10`: creates a 'Study Duration' plot
- `11`: creates an 'Expected Number of Subjects' plot
- `12`: creates an 'Analysis Times' plot
- `13`: creates a 'Cumulative Distribution Function' plot
- `14`: creates a 'Survival Function' plot

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

`theta` A vector of theta values.

`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

`showSource` If `TRUE`, the parameter names of the object will be printed which were used to create the plot; that may be, e.g., useful to check the values or to create own plots with `plot`.

`...` Optional `ggplot2` arguments.

Details

Generic function to plot all kinds of simulation results.

plot.StageResults *Stage Results Plotting*

Description

Plots the conditional power together with the likelihood function.

Usage

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

Arguments

x	The stage results at given stage, obtained from `getStageResults` or `getAnalysisResults`.
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: • `thetaRange`: A range of assumed effect sizes if testing means or a survival design was specified. Additionally, if testing means was selected, an assumed standard deviation can be specified (default is 1). • `piRange`: A range of assumed rates pi1 to calculate the conditional power. Additionally, if a two-sample comparison was selected, pi2 can be specified (default is the value from `getAnalysisResults`). • `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.
type	The plot type (default = 1). Note that at the moment only one type (the conditional power plot) is available.
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

`showSource` If `TRUE`, the parameter names of the object will be printed which were used to create the plot; that may be, e.g., useful to check the values or to create own plots with `plot`.

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(50, 51, 55),
    stDevs = c(130, 140, 120)
)

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

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

---

plot.TrialDesign *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(-1, 1, 0.01), nMax = NA_integer_,
  plotPointsEnabled = NA, legendPosition = NA_integer_,
  showSource = FALSE, ...)
```

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 'Boundaries' plot • 3: creates a 'Stage Levels' plot • 4: creates a 'Type One Error Spending' plot • 5: creates a 'Power and Early Stopping' plot • 6: creates an 'Average Sample Size and Power / Early Stop' plot • 7: creates an 'Power' plot • 8: creates an 'Early Stopping' plot • 9: creates an 'Average Sample Size' 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

showSource	If TRUE, the parameter names of the object will be printed which were used to create the plot; that may be, e.g., useful to check the values or to create own plots with plot.
...	Optional ggplot2 arguments.

Details

Generic function to plot a trial design.

Generic function to plot a trial design.

See Also

plot.TrialDesignSet 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.TrialDesignPlan

Trial Design Plan Plotting

Description

Plots a trial design plan.

Usage

```
## S3 method for class 'TrialDesignPlan'
plot(x, y, main = NA_character_,
  xlab = NA_character_, ylab = NA_character_,
  type = ifelse(x$.design$kMax == 1, 5, 1), palette = "Set1",
  theta = seq(-1, 1, 0.01), plotPointsEnabled = NA,
  legendPosition = NA_integer_, showSource = FALSE, ...)
```

Arguments

x	The trial design plan, obtained from getSampleSizeMeans, getSampleSizeRates, getSampleSizeSurvival, getPowerMeans, getPowerRates or getPowerSurvival.

`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 'Boundaries' plot
- `2`: creates a 'Boundaries Effect Scale' plot
- `3`: creates a 'Boundaries p Values Scale' plot
- `4`: creates a 'Type One Error Spending' plot
- `5`: creates a 'Sample Size' or 'Overall Power and Early Stopping' plot
- `6`: creates a 'Number of Events' or 'Sample Size' plot
- `7`: creates an 'Overall Power' plot
- `8`: creates an 'Overall Early Stopping' plot
- `9`: creates an 'Expected Number of Events' or 'Expected Sample Size' plot
- `10`: creates a 'Study Duration' plot
- `11`: creates an 'Expected Number of Subjects' plot
- `12`: creates an 'Analysis Times' plot
- `13`: creates a 'Cumulative Distribution Function' plot
- `14`: creates a 'Survival Function' plot

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

`theta` A vector of theta values.

`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

`showSource` If `TRUE`, the parameter names of the object will be printed which were used to create the plot; that may be, e.g., useful to check the values or to create own plots with `plot`.

`...` Optional `ggplot2` arguments.

Details

Generic function to plot all kinds of trial design plans.

`plot.TrialDesignSet` *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_,
  showSource = FALSE, ...)
```

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 'Boundaries' plot • `3`: creates a 'Stage Levels' plot • `4`: creates a 'Type One Error Spending' plot • `5`: creates a 'Power and Early Stopping' plot • `6`: creates an 'Average Sample Size and Power / Early Stop' plot • `7`: creates an 'Power' plot • `8`: creates an 'Early Stopping' plot • `9`: creates an 'Average Sample Size' 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

showSource — If TRUE, the parameter names of the object will be printed which were used to create the plot; that may be, e.g., useful to check the values or to create own plots with plot.

... — 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 = 4)

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

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 behavior 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 `read.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 behavior 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.

resetLogLevel *Reset Log Level*

Description

Resets the rpact log level.

Usage

```
resetLogLevel()
```

Details

This function is intended for debugging purposes only.

Examples

```
## Not run:
resetLogLevel()

## End(Not run)
```

rpact *RPACT - Confirmatory Adaptive Clinical Trial Design and Analysis*

Description

RPACT (R Package for Adaptive Clinical Trials) 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. It also comprises advanced methods for sample size calculations for fixed sample size designs incl., e.g., sample size calculation for survival trials with piecewise exponentially distributed survival times and staggered patients entry.

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 techniques of the adaptive methodology are also available, e.g., overall confidence intervals, overall p-values, and conditional and predictive power assessments. Simulations can be performed to assess the design characteristics of a (user-defined) sample size recalculation strategy. 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)

Gernot Wassmer, Friedrich Pahlke

References

Wassmer, G., Brannath, W. (2016) Group Sequential and Confirmatory Adaptive Designs in Clinical Trials (Springer Series in Pharmaceutical Statistics) <doi:10.1007/978-3-319-32562-0>

See Also

Useful links:

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

---

setLogLevel *Set Log Level*

Description

Sets the rpact log level.

Usage

```
setLogLevel(logLevel = c("PROGRESS", "ERROR", "WARN", "INFO", "DEBUG",
  "TRACE", "DISABLED"))
```

Arguments

logLevel	The new log level to set. Can be one of "PROGRESS", "ERROR", "WARN", "INFO", "DEBUG", "TRACE", "DISABLED".

Details

This function is intended for debugging purposes only.

Examples

```
## Not run:
setLogLevel("DEBUG")

## End(Not run)
```

SimulationResults *Class for Simulation Results*

Description

A class for simulation results.

Details

`SimulationResults` is the basic class for

- `SimulationResultsMeans`,
- `SimulationResultsRates`, and
- `SimulationResultsSurvival`.

SimulationResultsMeans *Class for Simulation Results Means*

Description

A class for simulation results means.

Details

Use `getSimulationMeans` to create an object of this type.

SimulationResultsRates *Class for Simulation Results Rates*

Description

A class for simulation results rates.

Details

Use `getSimulationRates` to create an object of this type.

SimulationResultsSurvival *Class for Simulation Results Survival*

Description

A class for simulation results survival.

Details

Use `getSimulationSurvival` to create an object of this type.

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.

testPackage *Test Package*

Description

These function allows the installed package rpact to be tested.

Usage

```
testPackage(outDir = ".", ..., completeUnitTestSetEnabled = TRUE,
  types = "tests", sourceDirectory = NULL)
```

Arguments

outDir	The output directory where all test results shall be saved. By default the current working directory is used.
completeUnitTestSetEnabled	If TRUE (default) all existing unit tests will be executed; a subset of all unit tests will be used otherwise.
types	The type(s) of tests to be done. Can be one or more of c("tests", "examples", "vignette Default is "tests" only.
sourceDirectory	An optional directory to look for .save files.

Details

This function creates the subdirectory rpact-tests in the specified output directory and copies all unit test files of the package to this newly created directory. Then the function runs all tests (or a subset of all tests if completeUnitTestSetEnabled is FALSE) using testInstalledPackage. The test results will be saved to the text file testthat.Rout that can be found in the subdirectory rpact-tests.

Examples

```
## Not run:
testPackage()

## End(Not run)
```

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.

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.

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.

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.

TrialDesignPlanSurvival_summary
Trial Design Plan Survival Set Summary

Description

Displays a summary of `TrialDesignPlanSurvival` object.

Usage

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

Details

Summarizes the parameters and results of a survival design.

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.

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.

`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.

`utilitiesForPiecewiseExponentialDistribution` *The Piecewise Exponential Distribution*

Description

Distribution function, quantile function and random number generation for the piecewise exponential distribution.

Usage

```
getPiecewiseExponentialDistribution(time, ...,
  piecewiseSurvivalTime = NA_real_, piecewiseLambda = NA_real_,
  kappa = 1)

ppwexp(t, ..., s = NA_real_, lambda = NA_real_, kappa = 1)

getPiecewiseExponentialQuantile(quantile, ...,
  piecewiseSurvivalTime = NA_real_, piecewiseLambda = NA_real_,
  kappa = 1)

qpwexp(q, ..., s = NA_real_, lambda = NA_real_, kappa = 1)

getPiecewiseExponentialRandomNumbers(n, ...,
  piecewiseSurvivalTime = NA_real_, piecewiseLambda = NA_real_,
  kappa = 1)

rpwexp(n, ..., s = NA_real_, lambda = NA_real_, kappa = 1)
```

Arguments

`...`
: Ensures that all arguments after `time` are be named and that a warning will be displayed if unknown arguments are passed.

`kappa`
: The kappa value. Is needed for the specification of the Weibull distribution. In this case, no piecewise definition is possible, i.e., only lambda and kappa need to be specified. This function is equivalent to pweibull(t, kappa, 1 / lambda) of the R core system, i.e., the scale parameter is 1 / 'hazard rate'. For example, getPiecewiseExponentialDistribution(time = 130, piecewiseLambda = 0.01, kappa = 4.2) and pweibull(q = 130, shape = 4.2, scale = 1 /0.01) provide the sample result.

`t, time` Vector of time values.

`s, piecewiseSurvivalTime`
Vector of start times defining the "time pieces".

`lambda, piecewiseLambda`
Vector of lambda values (hazard rates) corresponding to the start times.

`q, quantile` Vector of quantiles.

`n` Number of observations.

Details

`getPiecewiseExponentialDistribution` (short: `ppwexp`), `getPiecewiseExponentialQuantile` (short: `qpwexp`), and `getPiecewiseExponentialRandomNumbers` (short: `rpwexp`) provide probabilities, quantiles, and random numbers according to a piecewise exponential or a Weibull distribution. The piecewise definition is performed through a vector of starting times (`piecewiseSurvivalTime`) and a vector of hazard rates (`piecewiseLambda`). You can also use a list that defines the starting times and piecewise lambdas together and define piecewiseSurvivalTime as this list. The list needs to have the form, for example, piecewiseSurvivalTime <- list("0 - <6" = 0.025, "6 - <9" = 0.04, "9 - <15" = 0.015, ">=15" = 0.007) For the Weibull case, you can also specify a shape parameter kappa in order to calculated probabilities, quantiles, or random numbers. In this case, no piecewise definition is possible, i.e., only piecewiseLambda and kappa need to be specified.

Examples

```
# Calculate probabilties for a range of time values for a
# piecewise exponential distribution with hazard rates
# 0.025, 0.04, 0.015, and 0.007 in the intervals
# [0, 6), [6, 9), [9, 15), [15,Inf), respectively,
# and re-return the time values:
piecewiseSurvivalTime <- list(
    "0 - <6"   = 0.025,
    "6 - <9"   = 0.04,
    "9 - <15"  = 0.015,
    ">=15"     = 0.01)
y <- getPiecewiseExponentialDistribution(seq(0, 150, 15),
    piecewiseSurvivalTime = piecewiseSurvivalTime)
getPiecewiseExponentialQuantile(y,
    piecewiseSurvivalTime = piecewiseSurvivalTime)
```

---

utilitiesForSurvivalTrials

Survival Helper Functions for Conversion of Pi, Lambda, Median

---

Description

Functions to convert pi, lambda and median values into each other.

Usage

```
getLambdaByPi(piValue, eventTime = C_EVENT_TIME_DEFAULT, kappa = 1)

getLambdaByMedian(median, kappa = 1)

getHazardRatioByPi(pi1, pi2, eventTime = C_EVENT_TIME_DEFAULT,
  kappa = 1)

getPiByLambda(lambda, eventTime = C_EVENT_TIME_DEFAULT, kappa = 1)

getPiByMedian(median, eventTime = C_EVENT_TIME_DEFAULT, kappa = 1)

getMedianByLambda(lambda, kappa = 1)

getMedianByPi(piValue, eventTime = C_EVENT_TIME_DEFAULT, kappa = 1)
```

Arguments

`piValue, pi1, pi2, lambda, median`	Value that shall be converted.
`eventTime`	The assumed time under which the event rates are calculated, default is `12`.
`kappa`	The scale parameter of the Weibull distribution, default is `1`. The Weibull distribution cannot be used for the piecewise definition of the survival time distribution.

Details

Can be used, e.g., to convert median values into pi or lambda values for usage in `getSampleSizeSurvival` or `getPowerSurvival`.

`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 `write.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 behavior 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 write.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 behavior 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