Package ‘rpact’

August 24, 2022

Title Confirmatory Adaptive Clinical Trial Design and Analysis

Version 3.3.1

Date 2022-08-24

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 LGPL-3

Encoding UTF-8

LazyData true

URL `https://www.rpact.org`, `https://www.rpact.com`, `https://github.com/rpact-com/rpact`

BugReports `https://github.com/rpact-com/rpact/issues`

Language en-US

Depends R (>= 3.5.0)

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

LinkingTo Rcpp

Suggests parallel, ggplot2 (>= 2.2.0), testthat (>= 2.0.0), mnormt (>= 1.5-7), knitr (>= 1.19), rmarkdown (>= 1.10)

VignetteBuilder knitr, rmarkdown

RoxygenNote 7.2.1

Config/testthat/edition 3

Collate 'RcppExports.R' 'f_logger.R' 'f_core_constants.R' 'f_core_utilities.R' 'f_core_assertions.R' 'f_analysis_utilities.R' 'f_parameter_set_utilities.R' 'class_core_parameter_set.R' 'class_core_plot_settings.R' 'f_analysis_base.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_design_utilities.R' 'class_design_plan.R'
'class_design_power_and_asn.R' 'class_event_probabilities.R'
'f_simulation_utilities.R' 'f_simulation_base_survival.R'
'class_simulation_results.R' 'class_summary.R' 'data.R'
'f_analysis_base_means.R' 'f_analysis_base_rates.R'
'f_analysis_base_survival.R' 'f_analysis_enrichment.R'
'f_analysis_enrichment_means.R' 'f_analysis_enrichment_rates.R'
'f_analysis_enrichment_survival.R' 'f_analysis_multiarm.R'
'f_analysis_multiarm_means.R' 'f_analysis_multiarm_rates.R'
'f_analysis_multiarm_survival.R' 'f_core_output_formats.R'
'f_design_fisher_combination_test.R'
'f_design_group_sequential.R'
'f_design_sample_size_calculator.R' 'f_quality_assurance.R'
'f_simulation_base_means.R' 'f_simulation_base_rates.R'
'f_simulation_enrichment.R' 'f_simulation_enrichment_means.R'
'f_simulation_enrichment_rates.R'
'f_simulation_enrichment_survival.R' 'f_simulation_multiarm.R'
'f_simulation_multiarm_means.R' 'f_simulation_multiarm_rates.R'
'f_simulation_multiarm_survival.R' 'parameter_descriptions.R'
'pkgname.R'

NeedsCompilation yes

Author Gernot Wassmer [aut],
Friedrich Pahlke [aut, cre]

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

Repository CRAN

Date/Publication 2022-08-24 10:50:02 UTC

R topics documented:

`dataEnrichmentMeans` *Enrichment Dataset of Means*

Description

A dataset containing the sample sizes, means, and standard deviations of two groups. Use `getDataset(dataEnrichmentMean` to create a dataset object that can be processed by `getAnalysisResults`.

Usage

```
dataEnrichmentMeans
```

Format

A `data.frame` object.

`dataEnrichmentMeansStratified` *Stratified Enrichment Dataset of Means*

Description

A dataset containing the sample sizes, means, and standard deviations of two groups. Use `getDataset(dataEnrichmentMean` to create a dataset object that can be processed by `getAnalysisResults`.

Usage

```
dataEnrichmentMeansStratified
```

Format

A `data.frame` object.

`dataEnrichmentRates` *Enrichment Dataset of Rates*

Description

A dataset containing the sample sizes and events of two groups. Use `getDataset(dataEnrichmentRates)` to create a dataset object that can be processed by `getAnalysisResults`.

Usage

```
dataEnrichmentRates
```

Format

A `data.frame` object.

`dataEnrichmentRatesStratified`

Stratified Enrichment Dataset of Rates

Description

A dataset containing the sample sizes and events of two groups. Use `getDataset(dataEnrichmentRatesStratified)` to create a dataset object that can be processed by `getAnalysisResults`.

Usage

```
dataEnrichmentRatesStratified
```

Format

A `data.frame` object.

`dataEnrichmentSurvival`

Enrichment Dataset of Survival Data

Description

A dataset containing the log-rank statistics, events, and allocation ratios of two groups. Use `getDataset(dataEnrichmentSu`
to create a dataset object that can be processed by `getAnalysisResults`.

Usage

```
dataEnrichmentSurvival
```

Format

A `data.frame` object.

`dataEnrichmentSurvivalStratified`

Stratified Enrichment Dataset of Survival Data

Description

A dataset containing the log-rank statistics, events, and allocation ratios of two groups. Use `getDataset(dataEnrichmentSu`
to create a dataset object that can be processed by `getAnalysisResults`.

Usage

```
dataEnrichmentSurvivalStratified
```

Format

A `data.frame` object.

`dataMeans` *One-Arm Dataset of Means*

Description

A dataset containing the sample sizes, means, and standard deviations of one group. Use `getDataset(dataMeans)` to create a dataset object that can be processed by `getAnalysisResults`.

Usage

```
dataMeans
```

Format

A `data.frame` object.

`dataMultiArmMeans` *Multi-Arm Dataset of Means*

Description

A dataset containing the sample sizes, means, and standard deviations of four groups. Use `getDataset(dataMultiArmMeans` to create a dataset object that can be processed by `getAnalysisResults`.

Usage

```
dataMultiArmMeans
```

Format

A `data.frame` object.

`dataMultiArmRates` *Multi-Arm Dataset of Rates*

Description

A dataset containing the sample sizes and events of three groups. Use `getDataset(dataMultiArmRates)` to create a dataset object that can be processed by `getAnalysisResults`.

Usage

```
dataMultiArmRates
```

Format

A `data.frame` object.

`dataMultiArmSurvival` *Multi-Arm Dataset of Survival Data*

Description

A dataset containing the log-rank statistics, events, and allocation ratios of three groups. Use `getDataset(dataMultiArmSurvival)` to create a dataset object that can be processed by `getAnalysisResults`.

Usage

```
dataMultiArmSurvival
```

Format

A `data.frame` object.

`dataRates` *One-Arm Dataset of Rates*

Description

A dataset containing the sample sizes and events of one group. Use `getDataset(dataRates)` to create a dataset object that can be processed by `getAnalysisResults`.

Usage

```
dataRates
```

Format

A `data.frame` object.

`dataSurvival` *One-Arm Dataset of Survival Data*

Description

A dataset containing the log-rank statistics, events, and allocation ratios of one group. Use `getDataset(dataSurvival)` to create a dataset object that can be processed by `getAnalysisResults`.

Usage

```
dataSurvival
```

Format

A `data.frame` object.

`getAccrualTime` *Get Accrual Time*

Description

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

Usage

```
getAccrualTime(
  accrualTime = NA_real_,
  ...,
  accrualIntensity = NA_real_,
  accrualIntensityType = c("auto", "absolute", "relative"),
  maxNumberOfSubjects = NA_real_
)
```

Arguments

`accrualTime`	The assumed accrual time intervals for the study, default is `c(0, 12)` (for details see `getAccrualTime`).
`...`	Ensures that all arguments (starting from the "...") are to 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` (for details see `getAccrualTime`).
`accrualIntensityType`	A character value specifying the accrual intensity input type. Must be one of `"auto"`, `"absolute"`, or `"relative"`; default is `"auto"`, i.e., if all values are < 1 the type is `"relative"`, otherwise it is `"absolute"`.
`maxNumberOfSubjects`	The maximum number of subjects.

Value

Returns an `AccrualTime` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

Staggered patient entry

`accrualTime` is the time period of subjects' accrual in a study. It can be a value that defines the end of accrual or a vector. In this case, `accrualTime` can be used to define a non-constant accrual over time. For this, `accrualTime` is a vector that defines the accrual intervals. The first element of `accrualTime` must be equal to `0` and, additionally, `accrualIntensity` needs to be specified. `accrualIntensity` itself is a value or a vector (depending on the length of `accrualtime`) that defines the intensity how subjects enter the trial in the intervals defined through `accrualTime`.

`accrualTime` can also be a list that combines the definition of the accrual time and accrual intensity (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), `maxNumberOfSubjects > 0` needs to be specified and the end of accrual is calculated. In that case, `accrualIntensity` is the number of subjects per time unit, i.e., the absolute accrual intensity.

If the length of `accrualTime` equals the length of `accrualIntensity - 1` (i.e., the end of accrual is defined), `maxNumberOfSubjects` is calculated if the absolute accrual intensity is given. If all elements in `accrualIntensity` are smaller than 1, `accrualIntensity` defines the *relative* intensity how subjects enter the trial. For example, `accrualIntensity = c(0.1, 0.2)` specifies that in the second accrual interval the intensity is doubled as compared to the first accrual interval. The actual (absolute) accrual intensity is calculated for the calculated or given `maxNumberOfSubjects`. Note that the default is `accrualIntensity = 0.1` meaning that the *absolute* accrual intensity will be calculated.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

`getNumberOfSubjects` for calculating the number of subjects at given time points.

Examples

```
# Assume that in a trial the accrual after the first 6 months is doubled
# and the total accrual time is 30 months.
# Further assume that a total of 1000 subjects are entered in the trial.
# The number of subjects to be accrued in the first 6 months and afterwards
# is achieved through
getAccrualTime(accrualTime = c(0, 6, 30),
    accrualIntensity = c(0.1, 0.2), maxNumberOfSubjects = 1000)

# The same result is obtained via the list based definition
getAccrualTime(list(
     "0 - <6"   = 0.1,
     "6 - <=30" = 0.2),
```

```
     maxNumberOfSubjects = 1000)

# Calculate the end of accrual at given absolute intensity:
getAccrualTime(accrualTime = c(0, 6),
    accrualIntensity = c(18, 36), maxNumberOfSubjects = 1000)

# Via the list based definition this is
getAccrualTime(list(
    "0 - <6" = 18,
    ">=6" = 36),
    maxNumberOfSubjects = 1000)

# You can use an accrual time object in getSampleSizeSurvival() or
# getPowerSurvival().
# For example, if the maximum number of subjects and the follow up
# time needs to be calculated for a given effect size:
accrualTime = getAccrualTime(accrualTime = c(0, 6, 30),
    accrualIntensity = c(0.1, 0.2))
getSampleSizeSurvival(accrualTime = accrualTime, pi1 = 0.4, pi2 = 0.2)

# Or if the power and follow up time needs to be calculated for given
# number of events and subjects:
accrualTime = getAccrualTime(accrualTime = c(0, 6, 30),
    accrualIntensity = c(0.1, 0.2), maxNumberOfSubjects = 110)
getPowerSurvival(accrualTime = accrualTime, pi1 = 0.4, pi2 = 0.2,
maxNumberOfEvents = 46)

# How to show accrual time details

# You can use a sample size or power object as argument for the 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 = TRUE,
  thetaH0 = NA_real_,
  nPlanned = NA_real_,
  allocationRatioPlanned = 1,
  stage = NA_integer_,
  maxInformation = NULL,
  informationEpsilon = NULL
)
```

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` and should be created with the function `getDataset`. For more information see `getDataset`.
`...`	Further arguments to be passed to methods (cf. separate functions in "See Also" below), e.g., `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. For testing means, 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 exact 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 `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 is `1000`. `seed` Seed for simulating the power for Fisher's combination test. See above, default is a random seed. `intersectionTest` Defines the multiple test for the intersection hypotheses in the closed system of hypotheses when testing multiple hypotheses. Five options are available in multi-arm designs: `"Dunnett"`, `"Bonferroni"`, `"Simes"`, `"Sidak"`, and `"Hierarchical"`, default is `"Dunnett"`. Four options are available in population enrichment designs: `"SpiessensDebois"` (one subset only), `"Bonferroni"`, `"Simes"`, and `"Sidak"`, default is `"Simes"`.

	`varianceOption` Defines the way to calculate the variance in multiple treatment arms (> 2) or population enrichment designs for testing means. For multiple arms, three options are available: `"overallPooled"`, `"pairwisePooled"`, and `"notPooled"`, default is `"overallPooled"`. For enrichment designs, the options are: `"pooled"`, `"pooledFromFull"` (one subset only), and `"notPooled"`, default is `"pooled"`. `thetaH1` **and** `assumedStDevs` **or** `piTreatments`**,** `piControls` The assumed effect size or assumed rates to calculate the conditional power in multi-arm trials or enrichment designs. For survival designs, `thetaH1` refers to the hazard ratio. You can specify a value or a vector with elements referring to the treatment arms or the sub-populations, respectively. If not specified, the conditional power is calculated under the assumption of observed effect sizes, standard deviations, rates, or hazard ratios. `stratifiedAnalysis` For enrichment designs, typically a stratified analysis should be chosen. For testing means and rates, also a non-stratified analysis based on overall data can be performed. For survival data, only a stratified analysis is possible (see Brannath et al., 2009), default is `TRUE`.
`directionUpper`	Specifies the direction of the alternative, only applicable for one-sided testing; default is `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 (testing means and rates, respectively), it is `1` for the survival case (testing the hazard ratio). For non-inferiority designs, `thetaH0` is the non-inferiority bound. That is, in case of (one-sided) testing of • *means*: a value `!= 0` (or a value `!= 1` for testing the mean ratio) can be specified. • *rates*: a value `!= 0` (or a value `!= 1` for testing the risk ratio `pi1 / pi2`) can be specified. • *survival data*: a bound for testing H0: `hazard ratio = thetaH0 != 1` can be specified. 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`.
`nPlanned`	The additional (i.e., "new" and not cumulative) sample size planned for each of the subsequent stages. The argument must be a vector with length equal to the number of remaining stages and contain the combined sample size from both treatment groups if two groups are considered. For survival outcomes, it should contain the planned number of additional events. For multi-arm designs, it is the per-comparison (combined) sample size. For enrichment designs, it is the (combined) sample size for the considered sub-population.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`stage`	The stage number (optional). Default: total number of existing stages in the data input.

`maxInformation`	Positive integer value specifying the maximum information.
`informationEpsilon`	Positive integer value specifying the absolute information epsilon, which defines the maximum distance from the observed information to the maximum information that causes the final analysis. Updates at the final analysis in case the observed information at the final analysis is smaller ("under-running") than the planned maximum information `maxInformation`, default is 0. Alternatively, a floating-point number > 0 and < 1 can be specified to define a relative information epsilon.

Details

Given a design and a dataset, at given stage the function calculates the test results (effect sizes, stagewise 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.

For designs with more than two treatments arms (multi-arm designs) or enrichment designs a closed combination test is performed. That is, additionally the statistics to be used in a closed testing procedure are provided.

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. For the inverse normal combination test design with more than two stages, a warning informs that the validity of the confidence interval is theoretically shown only if no sample size change was performed.

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.

Final stage p-values, median unbiased effect estimates, and final confidence intervals are not calculated for multi-arm and enrichment designs.

Value

Returns an `AnalysisResults` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

Note on the dependency of `mnormt`

If `intersectionTest = "Dunnett"` or `intersectionTest = "SpiessensDebois"`, or the design is a conditional Dunnett design and the dataset is a multi-arm or enrichment dataset, `rpact` uses the R package mnormt to calculate the analysis results.

See Also

- `getObservedInformationRates` for recalculation the observed information rates.

Other analysis functions: `getClosedCombinationTestResults()`, `getClosedConditionalDunnettTestResults()`, `getConditionalPower()`, `getConditionalRejectionProbabilities()`, `getFinalConfidenceInterval()`, `getFinalPValue()`, `getRepeatedConfidenceIntervals()`, `getRepeatedPValues()`, `getStageResults()`, `getTestActions()`

Examples

```
# Example 1 One-Sample t Test
# Perform an analysis within a three-stage group sequential design with
# O'Brien & Fleming boundaries and one-sample data with a continuous outcome
# where H0: mu = 1.2 is to be tested

dsnGS <- getDesignGroupSequential()
dataMeans <- getDataset(
    n = c(30, 30),
    means = c(1.96, 1.76),
    stDevs = c(1.92, 2.01))
getAnalysisResults(design = dsnGS, dataInput = dataMeans, thetaH0 = 1.2)

# You can obtain the results when performing an inverse normal combination test
# with these data by using the commands

dsnIN <- getDesignInverseNormal()
getAnalysisResults(design = dsnIN, dataInput = dataMeans, thetaH0 = 1.2)

# Example 2 Use Function Approach with Time to Event Data
# Perform an analysis within a use function approach according to an
# O'Brien & Fleming type use function and survival data where
# where H0: hazard ratio = 1 is to be tested. The events were observed
# over time and maxInformation = 120, informationEpsilon = 5 specifies
# that 116 > 120 - 5 observed events defines the final analysis.

design <- getDesignGroupSequential(typeOfDesign = "asOF")
dataSurvival <- getDataset(
```

```
cumulativeEvents = c(33, 72, 116),
cumulativeLogRanks = c(1.33, 1.88, 1.902))
getAnalysisResults(design, dataInput = dataSurvival, maxInformation = 120,
informationEpsilon = 5)

# Example 3 Multi-Arm Design
# In a four-stage combination test design with O'Brien & Fleming boundaries
# at the first stage the second treatment arm was dropped. With the Bonferroni
# intersection test, the results together with the CRP, conditional power
# (assuming a total of 40 subjects for each comparison and effect sizes 0.5
# and 0.8 for treatment arm 1 and 3, respectively, and standard deviation 1.2),
# RCIs and p-values of a closed adaptive test procedure are
# obtained as follows with the given data (treatment arm 4 refers to the
# reference group; displayed with summary and plot commands):

data <- getDataset(
    n1 = c(22, 23),
    n2 = c(21, NA),
    n3 = c(20, 25),
    n4 = c(25, 27),
    means1 = c(1.63, 1.51),
    means2 = c(1.4, NA),
    means3 = c(0.91, 0.95),
    means4 = c(0.83, 0.75),
    stds1 = c(1.2, 1.4),
    stds2 = c(1.3, NA),
    stds3 = c(1.1, 1.14),
    stds4 = c(1.02, 1.18))

design <- getDesignInverseNormal(kMax = 4)
x <- getAnalysisResults(design, dataInput = data, intersectionTest = "Bonferroni",
    nPlanned = c(40, 40), thetaH1 = c(0.5, NA, 0.8), assumedStDevs = 1.2)
summary(x)
if (require(ggplot2)) plot(x, thetaRange = c(0, 0.8))

design <- getDesignConditionalDunnett(secondStageConditioning = FALSE)
y <- getAnalysisResults(design, dataInput = data,
    nPlanned = 40, thetaH1 = c(0.5, NA, 0.8), assumedStDevs = 1.2,  stage = 1)
summary(y)
if (require(ggplot2)) plot(y, thetaRange = c(0, 0.4))

# Example 4 Enrichment Design
# Perform an two-stage enrichment design analysis with O'Brien & Fleming boundaries
# where one sub-population (S1) and a full population (F) are considered as primary
# analysis sets. At interim, S1 is selected for further analysis and the sample
# size is increased accordingly. With the Spiessens & Debois intersection test,
# the results of a closed adaptive test procedure together with the CRP, repeated
# RCIs and p-values are obtained as follows with the given data (displayed with
# summary and plot commands):

design <- getDesignInverseNormal(kMax = 2, typeOfDesign = "OF")
dataS1 <- getDataset(
    means1 = c(13.2, 12.8),
```

```
    means2 = c(11.1, 10.8),
    stDev1 = c(3.4, 3.3),
    stDev2 = c(2.9, 3.5),
    n1 = c(21, 42),
    n2 = c(19, 39))
dataNotS1 <- getDataset(
    means1 = c(11.8, NA),
    means2 = c(10.5, NA),
    stDev1 = c(3.6, NA),
    stDev2 = c(2.7, NA),
    n1 = c(15, NA),
    n2 = c(13, NA))
dataBoth <- getDataset(S1 = dataS1, R = dataNotS1)

x <- getAnalysisResults(design, dataInput = dataBoth,
    intersectionTest = "SpiessensDebois",
    varianceOption = "pooledFromFull",
    stratifiedAnalysis = TRUE)
summary(x)
if (require(ggplot2)) plot(x, type = 2)
```

`getClosedCombinationTestResults`

Get Closed Combination Test Results

Description

Calculates and returns the results from the closed combination test in multi-arm and population enrichment designs.

Usage

```
getClosedCombinationTestResults(stageResults)
```

Arguments

`stageResults`	The results at given stage, obtained from `getStageResults`.

Value

Returns a `ClosedCombinationTestResults` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,

- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

Other analysis functions: `getAnalysisResults()`, `getClosedConditionalDunnettTestResults()`, `getConditionalPower()`, `getConditionalRejectionProbabilities()`, `getFinalConfidenceInterval()`, `getFinalPValue()`, `getRepeatedConfidenceIntervals()`, `getRepeatedPValues()`, `getStageResults()`, `getTestActions()`

Examples

```
# In a four-stage combination test design with O'Brien & Fleming boundaries
# at the first stage the second treatment arm was dropped. With the Bonferroni
# intersection test, the results of a closed adaptive test procedure are
# obtained as follows with the given data (treatment arm 4 refers to the
# reference group):
data <- getDataset(
    n1 = c(22, 23),
    n2 = c(21, NA),
    n3 = c(20, 25),
    n4 = c(25, 27),
    means1 = c(1.63, 1.51),
    means2 = c(1.4, NA),
    means3 = c(0.91, 0.95),
    means4 = c(0.83, 0.75),
    stds1 = c(1.2, 1.4),
    stds2 = c(1.3, NA),
    stds3 = c(1.1, 1.14),
    stds4 = c(1.02, 1.18))

design <- getDesignInverseNormal(kMax = 4)
stageResults <- getStageResults(design, dataInput = data,
    intersectionTest = "Bonferroni")
getClosedCombinationTestResults(stageResults)
```

`getClosedConditionalDunnettTestResults`

Get Closed Conditional Dunnett Test Results

Description

Calculates and returns the results from the closed conditional Dunnett test.

Usage

```
getClosedConditionalDunnettTestResults(
  stageResults,
  ...,
  stage = stageResults$stage
)
```

Arguments

`stageResults`	The results at given stage, obtained from `getStageResults`.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`stage`	The stage number (optional). Default: total number of existing stages in the data input.

Details

For performing the conditional Dunnett test the design must be defined through the function `getDesignConditionalDunnett` See Koenig et al. (2008) and Wassmer & Brannath (2016), chapter 11 for details of the test procedure.

Value

Returns a `ClosedCombinationTestResults` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

Other analysis functions: `getAnalysisResults()`, `getClosedCombinationTestResults()`, `getConditionalPower()`, `getConditionalRejectionProbabilities()`, `getFinalConfidenceInterval()`, `getFinalPValue()`, `getRepeatedConfidenceIntervals()`, `getRepeatedPValues()`, `getStageResults()`, `getTestActions()`

Examples

```
# In a two-stage design a conditional Dunnett test should be performed
# where the  unconditional second stage p-values should be used for the
# test decision.
# At the first stage the second treatment arm was dropped. The results of
# a closed conditionsal Dunnett test are obtained as follows with the given
# data (treatment arm 4 refers to the reference group):
data <- getDataset(
    n1 = c(22, 23),
    n2 = c(21, NA),
    n3 = c(20, 25),
    n4 = c(25, 27),
    means1 = c(1.63, 1.51),
    means2 = c(1.4, NA),
    means3 = c(0.91, 0.95),
    means4 = c(0.83, 0.75),
    stds1 = c(1.2, 1.4),
    stds2 = c(1.3, NA),
    stds3 = c(1.1, 1.14),
    stds4 = c(1.02, 1.18))

# For getting the results of the closed test procedure, use the following commands:
design <- getDesignConditionalDunnett(secondStageConditioning = FALSE)
stageResults <- getStageResults(design, dataInput = data)
getClosedConditionalDunnettTestResults(stageResults)
```

`getConditionalPower` *Get Conditional Power*

Description

Calculates and returns the conditional power.

Usage

```
getConditionalPower(stageResults, ..., nPlanned, allocationRatioPlanned = 1)
```

Arguments

`stageResults`	The results at given stage, obtained from `getStageResults`.
`...`	Further (optional) arguments to be passed: `thetaH1` **and** `assumedStDevs` **or** `piTreatments`**,** `piControl` The assumed effect size or assumed rates to calculate the conditional power in multi-arm trials or enrichment designs. For survival designs, `thetaH1` refers to the hazard ratio. You can specify a value or a vector with elements referring to the treatment arms or the sub-populations, respectively. For testing means, an assumed standard 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.
`nPlanned`	The additional (i.e., "new" and not cumulative) sample size planned for each of the subsequent stages. The argument must be a vector with length equal to the number of remaining stages and contain the combined sample size from both treatment groups if two groups are considered. For survival outcomes, it should contain the planned number of additional events. For multi-arm designs, it is the per-comparison (combined) sample size. For enrichment designs, it is the (combined) sample size for the considered sub-population.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.

Details

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

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

Value

Returns a `ConditionalPowerResults` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,

- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

`plot.StageResults` or `plot.AnalysisResults` for plotting the conditional power.

Other analysis functions: `getAnalysisResults()`, `getClosedCombinationTestResults()`, `getClosedConditionalDunn`
`getConditionalRejectionProbabilities()`, `getFinalConfidenceInterval()`, `getFinalPValue()`, `getRepeatedConfidenceIntervals()`, `getRepeatedPValues()`, `getStageResults()`, `getTestActions()`

Examples

```
data <- getDataset(
   n1     = c(22, 13, 22, 13),
   n2     = c(22, 11, 22, 11),
   means1 = c(1, 1.1, 1, 1),
   means2 = c(1.4, 1.5, 1, 2.5),
   stds1  = c(1, 2, 2, 1.3),
   stds2  = c(1, 2, 2, 1.3))
stageResults <- getStageResults(
   getDesignGroupSequential(kMax = 4),
   dataInput = data, stage = 2, directionUpper = FALSE)
getConditionalPower(stageResults, thetaH1 = -0.4,
   nPlanned = c(64, 64), assumedStDev = 1.5, allocationRatioPlanned = 3)
```

`getConditionalRejectionProbabilities`

Get Conditional Rejection Probabilities

Description

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

Usage

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

Arguments

`stageResults`	The results at given stage, obtained from `getStageResults`.
`...`	Further (optional) arguments to be passed: `iterations` Iterations for simulating the conditional rejection probabilities for Fisher's combination test. For checking purposes, it can be estimated via simulation with specified `iterations`. `seed` Seed for simulating the conditional rejection probabilities for Fisher's combination test. See above, default is a random seed.

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.

Value

Returns a `numeric` vector of length `kMax` or in case of multi-arm stage results a `matrix` (each column represents a stage, each row a comparison) containing the conditional rejection probabilities.

See Also

Other analysis functions: `getAnalysisResults()`, `getClosedCombinationTestResults()`, `getClosedConditionalDunn`
`getConditionalPower()`, `getFinalConfidenceInterval()`, `getFinalPValue()`, `getRepeatedConfidenceIntervals(`
`getRepeatedPValues()`, `getStageResults()`, `getTestActions()`

Examples

```
# Calculate CRP for a Fisher's combination test design with
# two remaining stages and check the results by simulation.
design <- getDesignFisher(kMax = 4,
    informationRates = c(0.1, 0.3, 0.8, 1), alpha = 0.01)
data <- getDataset(n = c(40, 40), events = c(20, 22))
sr <- getStageResults(design, data, thetaH0 = 0.4)
getConditionalRejectionProbabilities(sr)
getConditionalRejectionProbabilities(sr, simulateCRP = TRUE,
    seed = 12345, iterations = 10000)
```

`getData` *Get Simulation Data*

Description

Returns the aggregated simulation data.

Usage

```
getData(x)

getData.SimulationResults(x)
```

Arguments

`x`	A `SimulationResults` object created by `getSimulationMeans`, `getSimulationRates`, `getSimulationSurvival`, `getSimulationMultiArmMeans`, `getSimulationMultiArmRates`, or `getSimulationMultiArmSurvival`.

Details

This function can be used to get the aggregated simulated data from an simulation results object, for example, obtained by `getSimulationSurvival`. In this case, 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. `eventsNotAchieved`: 1 if number of events could not be reached with observed number of subjects, 0 otherwise.
13. `futilityPerStage`: 1 if study should be stopped for futility, 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. `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` or `pi1H1` and `pi2H1`.
17. `trialStop`: `TRUE` if study should be stopped for efficacy or futility or final stage, `FALSE` otherwise.
18. `hazardRatioEstimateLR`: The estimated hazard ratio, derived from the log-rank statistic.

A subset of variables is provided for `getSimulationMeans`, `getSimulationRates`, `getSimulationMultiArmMeans`, `getSimulationMultiArmRates`, or `getSimulationMultiArmSurvival`.

Value

Returns a `data.frame`.

Examples

```
results <- getSimulationSurvival(pi1 = seq(0.3,0.6,0.1), pi2 = 0.3, eventTime = 12,
    accrualTime = 24, plannedEvents = 40, maxNumberOfSubjects = 200,
    maxNumberOfIterations = 50)
data <- getData(results)
head(data)
dim(data)
```

`getDataset` *Get Dataset*

Description

Creates a dataset object and returns it.

Usage

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

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

Arguments

`...`	A `data.frame` or some data vectors defining the dataset.
`floatingPointNumbersEnabled`	If `TRUE`, sample sizes and event numbers can be specified as floating-point numbers (this make sense, e.g., for theoretical comparisons); by default `floatingPointNumbersEnabled` = `FALSE`, i.e., samples sizes and event numbers 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 stage-wise 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 stage-wise 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 stage-wise 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 stage-wise 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 stage-wise events, (one-sided) logrank statistics, and allocation ratios.
- An element of `DatasetMeans`, `DatasetRates`, and `DatasetSurvival` for more than one comparison is created by adding subsequent digits to the variable names. The system can analyze these data in a multi-arm many-to-one comparison setting where the group with the highest index represents the control group.

Prefix `overall[Capital case of first letter of variable name]...` for the variable names enables entering the overall (cumulative) results and calculates stage-wise statistics. Since rpact version 3.2, the prefix `cumulative[Capital case of first letter of variable name]...` or `cum[Capital case of first letter of variable name]...` can alternatively be used for this.

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

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

For multi-arm designs, the index refers to the considered comparison. For example,
`getDataset(events1=c(13, 33), logRanks1 = c(1.23, 1.55), events2 = c(16, NA), logRanks2 = c(1.55, NA))`
refers to the case where one active arm (1) is considered at both stages whereas active arm 2 was dropped at interim. Number of events and logrank statistics are entered for the corresponding comparison to control (see Examples).

For enrichment designs, the comparison of two samples is provided for an unstratified (sub-population wise) or stratified data input.
For unstratified (sub-population wise) data input the data sets are defined for the sub-populations S1, S2, ..., F, where F refers to the full populations. Use of `getDataset(S1 = , S2, ..., F = )` defines the data set to be used in `getAnalysisResults` (see examples)
For stratified data input the data sets are defined for the strata S1, S12, S2, ..., R, where R refers to the remainder of the strata such that the union of all sets is the full population. Use of `getDataset(S1 = , S12 = , S2, ..., R = )` defines the data set to be used in `getAnalysisResults` (see examples)
For survival data, for enrichment designs the log-rank statistics should be entered as stratified log-rank statistics in order to provide strong control of Type I error rate. For stratified data input, the variables to be specified in `getDataset()` are `events`, `expectedEvents`, `varianceEvents`, and `allocationRatios` or `overallEvents`, `overallExpectedEvents`, `overallVarianceEvents`, and `overallAllocationRatios`. From this, (stratified) log-rank tests are calculated.

Value

Returns a `Dataset` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

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(
    cumulativeSampleSizes = c(22, 33, 55, 66),
    cumulativeMeans = c(1.000, 1.033, 1.020, 1.017),
    cumulativeStDevs = 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(
    cumulativeSampleSizes1 = c(22, 33, 55, 66),
    cumulativeSampleSizes2 = c(22, 35, 57, 70),
    cumulativeMeans1  = c(1, 1.033, 1.020, 1.017),
    cumulativeMeans2  = c(1.4, 1.437, 2.040, 2.126),
    cumulativeStDevs1 = c(1, 1.38, 1.64, 1.58),
    cumulativeStDevs2 = 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 Means (three groups) where the comparison of
# treatment arm 1 to control is dropped at the second interim stage:
datasetOfMeans <- getDataset(
   cumN1      = c(22, 33, NA),
   cumN2      = c(20, 34, 56),
   cumN3      = c(22, 31, 52),
   cumMeans1  = c(1.64, 1.54, NA),
   cumMeans2  = c(1.7, 1.5, 1.77),
   cumMeans3  = c(2.5, 2.06, 2.99),
   cumStDevs1 = c(1.5, 1.9, NA),
   cumStDevs2 = c(1.3, 1.3, 1.1),
   cumStDevs3 = c(1, 1.3, 1.8))
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 Dataset of Rates (three groups) where the comparison of
# treatment arm 2 to control is dropped at the first interim stage:
datasetOfRates <- getDataset(
    cumN1      = c(22, 33, 44),
    cumN2      = c(20, NA, NA),
    cumN3      = c(20, 34, 44),
    cumEvents1 = c(11, 14, 22),
    cumEvents2 = c(17, NA, NA),
    cumEvents3 = c(17, 19, 33))
datasetOfRates

# Create a Survival Dataset
datasetSurvival <- getDataset(
    cumEvents = c(8, 15, 19, 31),
    cumAllocationRatios = c(1, 1, 1, 2),
    cumLogRanks = c(1.52, 1.98, 1.99, 2.11)
)
datasetSurvival

# Create a Survival Dataset with four comparisons where treatment
# arm 2 was dropped at the first interim stage, and treatment arm 4
# at the second.
datasetSurvival <- getDataset(
    cumEvents1   = c(18, 45, 56),
    cumEvents2   = c(22, NA, NA),
    cumEvents3   = c(12, 41, 56),
    cumEvents4   = c(27, 56, NA),
    cumLogRanks1 = c(1.52, 1.98, 1.99),
    cumLogRanks2 = c(3.43, NA, NA),
    cumLogRanks3 = c(1.45, 1.67, 1.87),
    cumLogRanks4 = c(1.12, 1.33, NA)
)
datasetSurvival

# Enrichment: Stratified and unstratified data input
# The following data are from one study. Only the first
# (stratified) data input enables a stratified analysis.

# Stratified data input
S1 <- getDataset(
    sampleSize1 = c(18, 17),
    sampleSize2 = c(12, 33),
    mean1       = c(125.6, 111.1),
    mean2       = c(107.7, 77.7),
    stDev1      = c(120.1, 145.6),
    stDev2      = c(128.5, 133.3))
S2 <- getDataset(
```

```
    sampleSize1 = c(11, NA),
    sampleSize2 = c(14, NA),
    mean1       = c(100.1, NA),
    mean2      = c( 68.3, NA),
    stDev1      = c(116.8, NA),
    stDev2      = c(124.0, NA))
S12 <- getDataset(
    sampleSize1 = c(21, 17),
    sampleSize2 = c(21, 12),
    mean1       = c(135.9, 117.7),
    mean2       = c(84.9, 107.7),
    stDev1      = c(185.0, 92.3),
    stDev2      = c(139.5, 107.7))
R <- getDataset(
    sampleSize1 = c(19, NA),
    sampleSize2 = c(33, NA),
    mean1       = c(142.4, NA),
    mean2       = c(77.1, NA),
    stDev1      = c(120.6, NA),
    stDev2      = c(163.5, NA))
dataEnrichment <- getDataset(S1 = S1, S2 = S2, S12 = S12, R = R)
dataEnrichment

# Unstratified data input
S1N <- getDataset(
    sampleSize1 = c(39, 34),
    sampleSize2 = c(33, 45),
    stDev1      = c(156.503, 120.084),
    stDev2      = c(134.025, 126.502),
    mean1       = c(131.146, 114.4),
    mean2       = c(93.191, 85.7))
S2N <- getDataset(
    sampleSize1 = c(32, NA),
    sampleSize2 = c(35, NA),
    stDev1      = c(163.645, NA),
    stDev2      = c(131.888, NA),
    mean1       = c(123.594, NA),
    mean2       = c(78.26, NA))
F <- getDataset(
    sampleSize1 = c(69, NA),
    sampleSize2 = c(80, NA),
    stDev1      = c(165.468, NA),
    stDev2      = c(143.979, NA),
    mean1       = c(129.296, NA),
    mean2       = c(82.187, NA))
dataEnrichmentN <- getDataset(S1 = S1N, S2 = S2N, F = F)
dataEnrichmentN
```

`getDesignCharacteristics` *Get Design Characteristics*

Description

Calculates the characteristics of a design and returns it.

Usage

```
getDesignCharacteristics(design)
```

Arguments

`design`	The trial 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. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

Other design functions: `getDesignConditionalDunnett()`, `getDesignFisher()`, `getDesignGroupSequential()`, `getDesignInverseNormal()`, `getGroupSequentialProbabilities()`, `getPowerAndAverageSampleNumber()`

Examples

```
# Calculate design characteristics for a three-stage O'Brien & Fleming
# design at power 90% and compare it with Pocock's design.
getDesignCharacteristics(getDesignGroupSequential(beta = 0.1))
getDesignCharacteristics(getDesignGroupSequential(beta = 0.1, typeOfDesign = "P"))
```

`getDesignConditionalDunnett`

Get Design Conditional Dunnett Test

Description

Defines the design to perform an analysis with the conditional Dunnett test.

Usage

```
getDesignConditionalDunnett(
  alpha = 0.025,
  informationAtInterim = 0.5,
  secondStageConditioning = TRUE
)
```

Arguments

`alpha`	The significance level alpha, default is `0.025`.
`informationAtInterim`	The information to be expected at interim, default is `informationAtInterim = 0.5`.
`secondStageConditioning`	The way the second stage p-values are calculated within the closed system of hypotheses. If `secondStageConditioning = FALSE` is specified, the unconditional adjusted p-values are used, otherwise conditional adjusted p-values are calculated, default is `secondStageConditioning = TRUE` (for details, see Koenig et al., 2008).

Details

For performing the conditional Dunnett test the design must be defined through this function. You can define the information fraction and the way of how to compute the second stage p-values only in the design definition, and not in the analysis call.
See `getClosedConditionalDunnettTestResults` for an example and Koenig et al. (2008) and Wassmer & Brannath (2016), chapter 11 for details of the test procedure.

Value

Returns a `TrialDesign` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

Other design functions: `getDesignCharacteristics()`, `getDesignFisher()`, `getDesignGroupSequential()`, `getDesignInverseNormal()`, `getGroupSequentialProbabilities()`, `getPowerAndAverageSampleNumber()`

`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("equalAlpha", "fullAlpha", "noInteraction", "userDefinedAlpha"),
  userAlphaSpending = NA_real_,
  alpha0Vec = NA_real_,
  informationRates = NA_real_,
  sided = 1,
  bindingFutility = NA,
  tolerance = 1e-14,
  iterations = 0,
  seed = NA_real_
)
```

Arguments

`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`kMax`	The maximum number of stages `K`, default is `3`. The maximum selectable `kMax` is `20` for group sequential or inverse normal and `6` for Fisher combination test designs.
`alpha`	The significance level alpha, default is `0.025`.
`method`	`"equalAlpha"`, `"fullAlpha"`, `"noInteraction"`, or `"userDefinedAlpha"`, default is `"equalAlpha"` (for details, see Wassmer, 1999).
`userAlphaSpending`	The user defined alpha spending. Numeric vector of length `kMax` containing the cumulative alpha-spending (Type I error rate) up to each interim stage: `0 <= alpha_1 <= ... <= alpha_K <= alpha`.
`alpha0Vec`	Stopping for futility bounds for stage-wise p-values.
`informationRates`	The 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 = TRUE` is specified the calculation of the critical values is affected by the futility bounds (default is `TRUE`).
`tolerance`	The numerical 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 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 `TrialDesign` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

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

Other design functions: `getDesignCharacteristics()`, `getDesignConditionalDunnett()`, `getDesignGroupSequential`
`getDesignInverseNormal()`, `getGroupSequentialProbabilities()`, `getPowerAndAverageSampleNumber()`

Examples

```
# Calculate critical values for a two-stage Fisher's combination test
# with full level alpha = 0.05 at the final stage and stopping for
# futility bound alpha0 = 0.50, as described in Bauer and Koehne (1994).
getDesignFisher(kMax = 2, method = "fullAlpha", alpha = 0.05, alpha0Vec = 0.50)
```

---

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 = 1L,
  informationRates = NA_real_,
  futilityBounds = NA_real_,
  typeOfDesign = c("OF", "P", "WT", "PT", "HP", "WToptimum", "asP", "asOF", "asKD",
    "asHSD", "asUser", "noEarlyEfficacy"),
  deltaWT = NA_real_,
  deltaPT1 = NA_real_,
  deltaPT0 = NA_real_,
  optimizationCriterion = c("ASNH1", "ASNIFH1", "ASNsum"),
  gammaA = NA_real_,
  typeBetaSpending = c("none", "bsP", "bsOF", "bsKD", "bsHSD", "bsUser"),
```

```
  userAlphaSpending = NA_real_,
  userBetaSpending = NA_real_,
  gammaB = NA_real_,
  bindingFutility = NA,
  betaAdjustment = NA,
  constantBoundsHP = 3,
  twoSidedPower = NA,
  delayedInformation = NA_real_,
  tolerance = 1e-08
)
```

Arguments

`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`kMax`	The maximum number of stages `K`, default is `3`. The maximum selectable `kMax` is `20` for group sequential or inverse normal and `6` for Fisher combination test designs.
`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`	Is the alternative one-sided (`1`) or two-sided (`2`), default is `1`.
`informationRates`	The information rates (that must be fixed prior to the trial), default is `(1:kMax) / kMax`.
`futilityBounds`	The futility bounds, defined on the test statistic z scale (numeric vector of length `kMax - 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"`), Pampallona & Tsiatis (`"PT"`), 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"`), no early efficacy stop (`"noEarlyEfficacy"`), default is `"OF"`.
`deltaWT`	Delta for Wang & Tsiatis Delta class.
`deltaPT1`	Delta1 for Pampallona & Tsiatis class rejecting H0 boundaries.
`deltaPT0`	Delta0 for Pampallona & Tsiatis class rejecting H1 boundaries.
`optimizationCriterion`	Optimization criterion for optimum design within Wang & Tsiatis class (`"ASNH1"`, `"ASNIFH1"`, `"ASNsum"`), default is `"ASNH1"`, see details.
`gammaA`	Parameter for alpha spending function.
`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"`, `"bsKD"`, `"bsHSD"`, `"bsUser"`, default is `"none"`).

`userAlphaSpending`	The user defined alpha spending. Numeric vector of length `kMax` containing the cumulative alpha-spending (Type I error rate) up to each interim stage: `0 <= alpha_1 <= ... <= alpha_K <= alpha`.
`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.
`bindingFutility`	If `bindingFutility = TRUE` is specified the calculation of the critical values is affected by the futility bounds and the futility threshold is binding in the sense that the study must be stopped if the futility condition was reached (default is `FALSE`).
`betaAdjustment`	For two-sided beta spending designs, if `betaAdjustement = TRUE` a linear adjustment of the beta spending values is performed if an overlapping of decision regions for futility stopping at earlier stages occurs, otherwise no adjustment is performed (default is `TRUE`).
`constantBoundsHP`	The constant bounds up to stage `kMax - 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.
`delayedInformation`	Delay of information for delayed response designs. Can be a numeric value or a numeric vector of length `kMax - 1`
`tolerance`	The numerical 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.

For optimum designs, `"ASNH1"` minimizes the expected sample size under H1, `"ASNIFH1"` minimizes the sum of the maximum sample and the expected sample size under H1, and `"ASNsum"` minimizes the sum of the maximum sample size, the expected sample size under a value midway H0 and H1, and the expected sample size under H1.

Value

Returns a `TrialDesign` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,

- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

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

Other design functions: `getDesignCharacteristics()`, `getDesignConditionalDunnett()`, `getDesignFisher()`, `getDesignInverseNormal()`, `getGroupSequentialProbabilities()`, `getPowerAndAverageSampleNumber()`

Examples

```
# Calculate two-sided critical values for a four-stage
# Wang & Tsiatis design with Delta = 0.25 at level alpha = 0.05
getDesignGroupSequential(kMax = 4, alpha = 0.05, sided = 2,
    typeOfDesign = "WT", deltaWT = 0.25)


# Calculate one-sided critical values and binding futility bounds for a three-stage
# design with alpha- and beta-spending functions according to Kim & DeMets with gamma = 2.5
# (planned informationRates as specified, default alpha = 0.025 and beta = 0.2)
getDesignGroupSequential(kMax = 3, informationRates = c(0.3, 0.75, 1),
    typeOfDesign = "asKD", gammaA = 2.5, typeBetaSpending = "bsKD",
    gammaB = 2.5, bindingFutility = TRUE)


# Calculate the Pocock type alpha spending critical values if the first
# interim analysis was performed after 40% of the maximum information was observed
# and the second after 40% of the maximum information was observed (default alpha = 0.025)
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 = 1L,
  informationRates = NA_real_,
  futilityBounds = NA_real_,
 typeOfDesign = c("OF", "P", "WT", "PT", "HP", "WToptimum", "asP", "asOF", "asKD",
    "asHSD", "asUser", "noEarlyEfficacy"),
  deltaWT = NA_real_,
  deltaPT1 = NA_real_,
  deltaPT0 = NA_real_,
  optimizationCriterion = c("ASNH1", "ASNIFH1", "ASNsum"),
  gammaA = NA_real_,
  typeBetaSpending = c("none", "bsP", "bsOF", "bsKD", "bsHSD", "bsUser"),
  userAlphaSpending = NA_real_,
  userBetaSpending = NA_real_,
  gammaB = NA_real_,
  bindingFutility = NA,
  betaAdjustment = NA,
  constantBoundsHP = 3,
  twoSidedPower = NA,
  tolerance = 1e-08
)
```

Arguments

`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`kMax`	The maximum number of stages `K`, default is `3`. The maximum selectable `kMax` is `20` for group sequential or inverse normal and `6` for Fisher combination test designs.
`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`	Is the alternative one-sided (`1`) or two-sided (`2`), default is `1`.
`informationRates`	The information rates (that must be fixed prior to the trial), default is `(1:kMax) / kMax`.
`futilityBounds`	The futility bounds, defined on the test statistic z scale (numeric vector of length `kMax - 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"`), Pampallona & Tsiatis (`"PT"`), 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"`), no early efficacy stop (`"noEarlyEfficacy"`), default is `"OF"`.
`deltaWT`	Delta for Wang & Tsiatis Delta class.
`deltaPT1`	Delta1 for Pampallona & Tsiatis class rejecting H0 boundaries.
`deltaPT0`	Delta0 for Pampallona & Tsiatis class rejecting H1 boundaries.
`optimizationCriterion`	Optimization criterion for optimum design within Wang & Tsiatis class (`"ASNH1"`, `"ASNIFH1"`, `"ASNsum"`), default is `"ASNH1"`, see details.
`gammaA`	Parameter for alpha spending function.
`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"`, `"bsKD"`, `"bsHSD"`, `"bsUser"`, default is `"none"`).
`userAlphaSpending`	The user defined alpha spending. Numeric vector of length `kMax` containing the cumulative alpha-spending (Type I error rate) up to each interim stage: `0 <= alpha_1 <= ... <= alpha_K <= alpha`.
`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.
`bindingFutility`	If `bindingFutility = TRUE` is specified the calculation of the critical values is affected by the futility bounds and the futility threshold is binding in the sense that the study must be stopped if the futility condition was reached (default is `FALSE`).
`betaAdjustment`	For two-sided beta spending designs, if `betaAdjustement = TRUE` a linear adjustment of the beta spending values is performed if an overlapping of decision regions for futility stopping at earlier stages occurs, otherwise no adjustment is performed (default is `TRUE`).
`constantBoundsHP`	The constant bounds up to stage `kMax - 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 numerical 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.

For optimum designs, `"ASNH1"` minimizes the expected sample size under H1, `"ASNIFH1"` minimizes the sum of the maximum sample and the expected sample size under H1, and `"ASNsum"` minimizes the sum of the maximum sample size, the expected sample size under a value midway H0 and H1, and the expected sample size under H1.

Value

Returns a `TrialDesign` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

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

Other design functions: `getDesignCharacteristics()`, `getDesignConditionalDunnett()`, `getDesignFisher()`, `getDesignGroupSequential()`, `getGroupSequentialProbabilities()`, `getPowerAndAverageSampleNumber()`

Examples

```
# Calculate two-sided critical values for a four-stage
# Wang & Tsiatis design with Delta = 0.25 at level alpha = 0.05
getDesignInverseNormal(kMax = 4, alpha = 0.05, sided = 2,
    typeOfDesign = "WT", deltaWT = 0.25)

# Defines a two-stage design at one-sided alpha = 0.025 with provision of early stopping
# if the one-sided p-value exceeds 0.5 at interim and no early stopping for efficacy.
# The futility bound is non-binding.
getDesignInverseNormal(kMax = 2, typeOfDesign = "noEarlyEfficacy", futilityBounds = 0)


# Calculate one-sided critical values and binding futility bounds for a three-stage
# design with alpha- and beta-spending functions according to Kim & DeMets with gamma = 2.5
# (planned informationRates as specified, default alpha = 0.025 and beta = 0.2)
```

```
getDesignInverseNormal(kMax = 3, informationRates = c(0.3, 0.75, 1),
    typeOfDesign = "asKD", gammaA = 2.5, typeBetaSpending = "bsKD",
    gammaB = 2.5, bindingFutility = TRUE)
```

`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, you need to specify the variable `variedParameters`).

Details

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

Value

Returns a `TrialDesignSet` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `length` to obtain the number of design,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

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, type = 1)


# Example 3 (use of designs instead of design)
d1 <- getDesignGroupSequential(
    alpha = 0.05, kMax = 2,
    sided = 1, beta = 0.2, typeOfDesign = "asHSD",
    gammaA = 0.5, typeBetaSpending = "bsHSD", gammaB = 0.5
)
d2 <- getDesignGroupSequential(
    alpha = 0.05, kMax = 4,
    sided = 1, beta = 0.2, typeOfDesign = "asP",
    typeBetaSpending = "bsP"
)
designSet <- getDesignSet(
    designs = c(d1, d2),
    variedParameters = c("typeOfDesign", "kMax")
)

if (require(ggplot2)) plot(designSet, type = 8, nMax = 20)
```

`getEventProbabilities` *Get Event Probabilities*

Description

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

Usage

```
getEventProbabilities(
  time,
  ...,
  accrualTime = c(0, 12),
  accrualIntensity = 0.1,
  accrualIntensityType = c("auto", "absolute", "relative"),
  kappa = 1,
  piecewiseSurvivalTime = NA_real_,
  lambda2 = NA_real_,
  lambda1 = NA_real_,
  allocationRatioPlanned = 1,
  hazardRatio = NA_real_,
  dropoutRate1 = 0,
  dropoutRate2 = 0,
  dropoutTime = 12,
  maxNumberOfSubjects = NA_real_
)
```

Arguments

`time`	A numeric vector with time values.
`...`	Ensures that all arguments (starting from the "...") are to 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)` (for details see `getAccrualTime`).
`accrualIntensity`	A vector of accrual intensities, default is the relative intensity `0.1` (for details see `getAccrualTime`).
`accrualIntensityType`	A character value specifying the accrual intensity input type. Must be one of `"auto"`, `"absolute"`, or `"relative"`; default is `"auto"`, i.e., if all values are < 1 the type is `"relative"`, otherwise it is `"absolute"`.
`kappa`	A numeric value > 0. A `kappa != 1` will be used for the specification of the shape of the Weibull distribution. Default is `1`, i.e., the exponential survival distribution is used instead of the Weibull distribution. Note that the Weibull distribution cannot be used for the piecewise definition of the survival time distribution, i.e., only `piecewiselambda` (as a single value) and `kappa` can be specified. This

	function is equivalent to `pweibull(t, shape = kappa, scale = 1 / lambda)` of the `stats` package, 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.
`piecewiseSurvivalTime`	A vector that specifies the time intervals for the piecewise definition of the exponential survival time cumulative distribution function (for details see `getPiecewiseSurvivalTime`).
`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 `n1 / n2` 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.
`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, there is no default.
`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

The function computes the overall event probabilities in a two treatment groups design. For details of the parameters see `getSampleSizeSurvival`.

Value

Returns a `EventProbabilities` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

Examples

```
# Calculate event probabilities for staggered subjects' entry, piecewisely defined
# survival time and hazards, and plot it.
timeVector <- seq(0, 100, 1)
y <- getEventProbabilities(timeVector, accrualTime = c(0, 20, 60),
    accrualIntensity = c(5, 20),
    piecewiseSurvivalTime = c(0, 20, 80),
    lambda2 = c(0.02, 0.06, 0.1),
    hazardRatio = 2
)

plot(timeVector, y$cumulativeEventProbabilities, type = 'l')
```

`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,
  ...,
  directionUpper = TRUE,
  thetaH0 = NA_real_,
  tolerance = 1e-06,
  stage = NA_integer_
)
```

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` and should be created with the function `getDataset`. For more information see `getDataset`.
`...`	Further (optional) arguments to be passed: `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 exact 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 `TRUE`.
`directionUpper`	Specifies the direction of the alternative, only applicable for one-sided testing; default is `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 (testing means and rates, respectively), it is `1` for the survival case (testing the hazard ratio). For non-inferiority designs, `thetaH0` is the non-inferiority bound. That is, in case of (one-sided) testing of • *means*: a value `!= 0` (or a value `!= 1` for testing the mean ratio) can be specified. • *rates*: a value `!= 0` (or a value `!= 1` for testing the risk ratio `pi1 / pi2`) can be specified. • *survival data*: a bound for testing H0: `hazard ratio = thetaH0 != 1` can be specified. 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`.
`tolerance`	The numerical tolerance, default is `1e-06`.
`stage`	The stage number (optional). Default: total number of existing stages in the data input.

Details

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

For the inverse normal combination test design with more than two stages, a warning informs that the validity of the confidence interval is theoretically shown only if no sample size change was performed.

Value

Returns a `list` containing

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

See Also

Other analysis functions: `getAnalysisResults()`, `getClosedCombinationTestResults()`, `getClosedConditionalDunn`
`getConditionalPower()`, `getConditionalRejectionProbabilities()`, `getFinalPValue()`, `getRepeatedConfidence`
`getRepeatedPValues()`, `getStageResults()`, `getTestActions()`

Examples

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

`getFinalPValue` *Get Final P Value*

Description

Returns the final p-value for given stage results.

Usage

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

Arguments

`stageResults`	The results at given stage, obtained from `getStageResults`.
`...`	Only available for backward compatibility.

Details

The calculation of the final p-value is based on the stage-wise 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.

Value

Returns a `list` containing

- `finalStage`,
- `pFinal`.

See Also

Other analysis functions: `getAnalysisResults()`, `getClosedCombinationTestResults()`, `getClosedConditionalDunn` `getConditionalPower()`, `getConditionalRejectionProbabilities()`, `getFinalConfidenceInterval()`, `getRepeatedConfidenceIntervals()`, `getRepeatedPValues()`, `getStageResults()`, `getTestActions()`

Examples

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

`getGroupSequentialProbabilities`

Get Group Sequential Probabilities

Description

Calculates probabilities in the group sequential setting.

Usage

```
getGroupSequentialProbabilities(decisionMatrix, informationRates)
```

Arguments

`decisionMatrix`	A matrix with either 2 or 4 rows and kMax = length(informationRates) columns, see details.
`informationRates`	The information rates (that must be fixed prior to the trial), default is `(1:kMax) / kMax`.

Details

Given a sequence of information rates (fixing the correlation structure), and decisionMatrix with either 2 or 4 rows and kMax = length(informationRates) columns, this function calculates a probability matrix containing, for two rows, the probabilities:
P(Z_1 <- l_1), P(l_1 <- Z_1 < u_1, Z_2 < l_1),..., P(l_kMax-1 <- Z_kMax-1 < u_kMax-1, Z_kMax < l_l_kMax)
P(Z_1 <- u_1), P(l_1 <- Z_1 < u_1, Z_2 < u_1),..., P(l_kMax-1 <- Z_kMax-1 < u_kMax-1, Z_kMax < u_l_kMax)
P(Z_1 <- Inf), P(l_1 <- Z_1 < u_1, Z_2 < Inf),..., P(l_kMax-1 <- Z_kMax-1 < u_kMax-1, Z_kMax < Inf)
with continuation matrix
l_1,...,l_kMax
u_1,...,u_kMax
For 4 rows, the continuation region contains of two regions and the probability matrix is obtained analogously (cf., Wassmer and Brannath, 2016).

See Also

Other design functions: `getDesignCharacteristics()`, `getDesignConditionalDunnett()`, `getDesignFisher()`, `getDesignGroupSequential()`, `getDesignInverseNormal()`, `getPowerAndAverageSampleNumber()`

Examples

```
# Calculate Type I error rates in the two-sided group sequential setting when
# performing kMax interim stages with constant critical boundaries at level alpha:
alpha <- 0.05
kMax <- 10
decisionMatrix <- matrix(c(
    rep(-qnorm(1 - alpha / 2), kMax),
    rep(qnorm(1 - alpha / 2), kMax)
), nrow = 2, byrow = TRUE)
informationRates <- (1:kMax) / kMax
probs <- getGroupSequentialProbabilities(decisionMatrix, informationRates)
cumsum(probs[3, ] - probs[2, ] + probs[1, ])

# Do the same for a one-sided design without futility boundaries:
decisionMatrix <- matrix(c(
    rep(-Inf, kMax),
    rep(qnorm(1 - alpha), kMax)
), nrow = 2, byrow = TRUE)
informationRates <- (1:kMax) / kMax
probs <- getGroupSequentialProbabilities(decisionMatrix, informationRates)
cumsum(probs[3, ] - probs[2, ])

# Check that two-sided Pampallona and Tsiatis boundaries with binding
# futility bounds obtain Type I error probabilities equal to alpha:
x <- getDesignGroupSequential(
    alpha = 0.05, beta = 0.1, kMax = 3, typeOfDesign = "PT",
    deltaPT0 = 0, deltaPT1 = 0.4, sided = 2, bindingFutility = TRUE
)
dm <- matrix(c(
```

```
    -x$criticalValues, -x$futilityBounds, 0,
    x$futilityBounds, 0, x$criticalValues
), nrow = 4, byrow = TRUE)
dm[is.na(dm)] <- 0
probs <- getGroupSequentialProbabilities(
    decisionMatrix = dm, informationRates = (1:3) / 3
)
sum(probs[5, ] - probs[4, ] + probs[1, ])

# Check the Type I error rate decrease when using non-binding futility bounds:
x <- getDesignGroupSequential(
    alpha = 0.05, beta = 0.1, kMax = 3, typeOfDesign = "PT",
    deltaPT0 = 0, deltaPT1 = 0.4, sided = 2, bindingFutility = FALSE
)
dm <- matrix(c(
    -x$criticalValues, -x$futilityBounds, 0,
    x$futilityBounds, 0, x$criticalValues
), nrow = 4, byrow = TRUE)
dm[is.na(dm)] <- 0
probs <- getGroupSequentialProbabilities(
    decisionMatrix = dm, informationRates = (1:3) / 3
)
sum(probs[5, ] - probs[4, ] + probs[1, ])
```

`getNumberOfSubjects` *Get Number Of Subjects*

Description

Returns the number of recruited subjects at given time vector.

Usage

```
getNumberOfSubjects(
  time,
  ...,
  accrualTime = c(0, 12),
  accrualIntensity = 0.1,
  accrualIntensityType = c("auto", "absolute", "relative"),
  maxNumberOfSubjects = NA_real_
)
```

Arguments

`time`	A numeric vector with time values.
`...`	Ensures that all arguments (starting from the "...") are to 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)` (for details see `getAccrualTime`).
`accrualIntensity`	A vector of accrual intensities, default is the relative intensity `0.1` (for details see `getAccrualTime`).
`accrualIntensityType`	A character value specifying the accrual intensity input type. Must be one of `"auto"`, `"absolute"`, or `"relative"`; default is `"auto"`, i.e., if all values are < 1 the type is `"relative"`, otherwise it is `"absolute"`.
`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

Calculate number of subjects over time range at given accrual time vector and accrual intensity. Intensity can either be defined in absolute or relative terms (for the latter, `maxNumberOfSubjects` needs to be defined)
The function is used by `getSampleSizeSurvival`.

Value

Returns a `NumberOfSubjects` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

`AccrualTime` for defining the accrual time.

Examples

```
getNumberOfSubjects(time = seq(10, 70, 10), accrualTime = c(0, 20, 60),
    accrualIntensity = c(5, 20))

getNumberOfSubjects(time = seq(10, 70, 10), accrualTime = c(0, 20, 60),
    accrualIntensity = c(0.1, 0.4), maxNumberOfSubjects = 900)
```

`getObservedInformationRates` *Get Observed Information Rates*

Description

Recalculates the observed information rates from the specified dataset.

Usage

```
getObservedInformationRates(
  dataInput,
  ...,
  maxInformation = NULL,
  informationEpsilon = NULL,
  stage = NA_integer_
)
```

Arguments

`dataInput`	The dataset for which the information rates shall be recalculated.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`maxInformation`	Positive integer value specifying the maximum information.
`informationEpsilon`	Positive integer value specifying the absolute information epsilon, which defines the maximum distance from the observed information to the maximum information that causes the final analysis. Updates at the final analysis in case the observed information at the final analysis is smaller ("under-running") than the planned maximum information `maxInformation`, default is 0. Alternatively, a floating-point number > 0 and < 1 can be specified to define a relative information epsilon.
`stage`	The stage number (optional). Default: total number of existing stages in the data input.

Details

For means and rates the maximum information is the maximum number of subjects or the relative proportion if `informationEpsilon` < 1; for survival data it is the maximum number of events or the relative proportion if `informationEpsilon` < 1.

See Also

- getAnalysisResults for using `getObservedInformationRates` implicit,
- https://www.rpact.com/vignettes/rpact_boundary_update_example

Examples

```
# Absolute information epsilon:
# decision rule 45 >= 46 - 1, i.e., under-running
data <- getDataset(
    overallN = c(22, 45),
    overallEvents = c(11, 28)
)
getObservedInformationRates(data,
    maxInformation = 46, informationEpsilon = 1
)

# Relative information epsilon:
# last information rate = 45/46 = 0.9783,
# is > 1 - 0.03 = 0.97, i.e., under-running
data <- getDataset(
    overallN = c(22, 45),
    overallEvents = c(11, 28)
)
getObservedInformationRates(data,
    maxInformation = 46, informationEpsilon = 0.03
)
```

`getOutputFormat` *Get Output Format*

Description

With this function the format of the standard outputs of all `rpact` objects can be shown and written to a file.

Usage

```
getOutputFormat(
  parameterName = NA_character_,
  ...,
  file = NA_character_,
  default = FALSE,
  fields = TRUE
)
```

Arguments

`parameterName`	The name of the parameter whose output format shall be returned. Leave the default `NA_character_` if the output format of all parameters shall be returned.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`file`	An optional file name where to write the output formats (see Details for more information).
`default`	If `TRUE` the default output format of the specified parameter(s) will be returned, default is `FALSE`.
`fields`	If `TRUE` the names of all affected object fields will be displayed, default is `TRUE`.

Details

Output formats can be written to a text file by specifying a `file`. See `setOutputFormat`() to learn how to read a formerly saved file.

Note that the `parameterName` must not match exactly, e.g., for p-values the following parameter names will be recognized amongst others:

1. `p value`
2. `p.values`
3. `p-value`
4. `pValue`
5. `rpact.output.format.p.value`

Value

A named list of output formats.

See Also

Other output formats: `setOutputFormat()`

Examples

```
# show output format of p values
getOutputFormat("p.value")

# set new p value output format
setOutputFormat("p.value", digits = 5, nsmall = 5)

# show sample sizes as smallest integers not less than the not rounded values
setOutputFormat("sample size", digits = 0, nsmall = 0, roundFunction = "ceiling")
getSampleSizeMeans()

# show sample sizes as smallest integers not greater than the not rounded values
setOutputFormat("sample size", digits = 0, nsmall = 0, roundFunction = "floor")
getSampleSizeMeans()
```

```
# set new sample size output format without round function
setOutputFormat("sample size", digits = 2, nsmall = 2)
getSampleSizeMeans()

# reset sample size output format to default
setOutputFormat("sample size")
getSampleSizeMeans()
getOutputFormat("sample size")
```

`getPiecewiseSurvivalTime`

Get Piecewise Survival Time

Description

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

Usage

```
getPiecewiseSurvivalTime(
  piecewiseSurvivalTime = NA_real_,
  ...,
  lambda1 = NA_real_,
  lambda2 = NA_real_,
  hazardRatio = NA_real_,
  pi1 = NA_real_,
  pi2 = NA_real_,
  median1 = NA_real_,
  median2 = NA_real_,
  eventTime = 12,
  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 (starting from the "...") are to 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, there is no default.
`pi1`	A numeric value or vector that represents the assumed event rate in the treatment group, default is `seq(0.2, 0.5, 0.1)` (power calculations and simulations) or `seq(0.4, 0.6, 0.1)` (sample size calculations).
`pi2`	A numeric value that represents the assumed event rate in the control group, default is `0.2`.
`median1`	The assumed median survival time in the treatment group, there is no default.
`median2`	The assumed median survival time in the reference group, there is no default.
`eventTime`	The assumed time under which the event rates are calculated, default is `12`.
`kappa`	A numeric value > 0. A `kappa != 1` will be used for the specification of the shape of the Weibull distribution. Default is `1`, i.e., the exponential survival distribution is used instead of the Weibull distribution. Note that the Weibull distribution cannot be used for the piecewise definition of the survival time distribution, i.e., only `piecewiselambda` (as a single value) and `kappa` can be specified. This function is equivalent to `pweibull(t, shape = kappa, scale = 1 / lambda)` of the `stats` package, 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.
`delayedResponseAllowed`	If `TRUE`, delayed response is allowed; otherwise it will be validated that the response is not delayed, default is `FALSE`.

Value

Returns a `PiecewiseSurvivalTime` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

Piecewise survival time

The first element of the vector `piecewiseSurvivalTime` 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).

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

Examples

```
getPiecewiseSurvivalTime(lambda2 = 0.5, hazardRatio = 0.8)

getPiecewiseSurvivalTime(lambda2 = 0.5, lambda1 = 0.4)

getPiecewiseSurvivalTime(pi2 = 0.5, hazardRatio = 0.8)

getPiecewiseSurvivalTime(pi2 = 0.5, pi1 = 0.4)

getPiecewiseSurvivalTime(pi1 = 0.3)

getPiecewiseSurvivalTime(hazardRatio = c(0.6, 0.8), lambda2 = 0.4)

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

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 <- 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
# getSampleSizeSurvival():
getSampleSizeSurvival(piecewiseSurvivalTime = pwst)

# The object created by getPiecewiseSurvivalTime() can be used directly in
# getPowerSurvival():
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 trial design.
`theta`	A vector of standardized effect sizes (theta values), default is a sequence from -1 to 1.
`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. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

Other design functions: `getDesignCharacteristics()`, `getDesignConditionalDunnett()`, `getDesignFisher()`, `getDesignGroupSequential()`, `getDesignInverseNormal()`, `getGroupSequentialProbabilities()`

Examples

```
# Calculate power, stopping probabilities, and expected sample
# size for the default design with specified theta and nMax
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,
  ...,
  groups = 2L,
  normalApproximation = FALSE,
  meanRatio = FALSE,
  thetaH0 = ifelse(meanRatio, 1, 0),
  alternative = seq(0, 1, 0.2),
  stDev = 1,
  directionUpper = NA,
  maxNumberOfSubjects = NA_real_,
  allocationRatioPlanned = NA_real_
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to 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`.
`normalApproximation`	The type of computation of the p-values. If `TRUE`, the variance is assumed to be known, default is `FALSE`, i.e., the calculations are performed with the t distribution.
`meanRatio`	If `TRUE`, the sample size for one-sided testing of H0: `mu1 / mu2 = thetaH0` is calculated, default is `FALSE`.

`thetaH0`	The null hypothesis value, default is `0` for the normal and the binary case (testing means and rates, respectively), it is `1` for the survival case (testing the hazard ratio). For non-inferiority designs, `thetaH0` is the non-inferiority bound. That is, in case of (one-sided) testing of • *means*: a value `!= 0` (or a value `!= 1` for testing the mean ratio) can be specified. • *rates*: a value `!= 0` (or a value `!= 1` for testing the risk ratio `pi1 / pi2`) can be specified. • *survival data*: a bound for testing H0: `hazard ratio = thetaH0 != 1` can be specified. 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`.
`alternative`	The alternative hypothesis value for testing means. This can be a vector of assumed alternatives, default is `seq(0, 1, 0.2)` (power calculations) or `seq(0.2, 1, 0.2)` (sample size calculations).
`stDev`	The standard deviation under which the sample size or power calculation is performed, 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` which means that larger values of the test statistics yield smaller p-values.
`maxNumberOfSubjects`	`maxNumberOfSubjects > 0` needs to be specified. For two treatment arms, it is the maximum number of subjects for both treatment arms.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.

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 `TrialDesignPlan` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,

- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

Other power functions: `getPowerRates()`, `getPowerSurvival()`

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,
  ...,
  groups = 2L,
```

```
  riskRatio = FALSE,
  thetaH0 = ifelse(riskRatio, 1, 0),
  pi1 = seq(0.2, 0.5, 0.1),
  pi2 = 0.2,
  directionUpper = NA,
  maxNumberOfSubjects = NA_real_,
  allocationRatioPlanned = NA_real_
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to 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 `TRUE`, the power for one-sided testing of H0: `pi1 / pi2 = thetaH0` is calculated, default is `FALSE`.
`thetaH0`	The null hypothesis value, default is `0` for the normal and the binary case (testing means and rates, respectively), it is `1` for the survival case (testing the hazard ratio). For non-inferiority designs, `thetaH0` is the non-inferiority bound. That is, in case of (one-sided) testing of • *means*: a value `!= 0` (or a value `!= 1` for testing the mean ratio) can be specified. • *rates*: a value `!= 0` (or a value `!= 1` for testing the risk ratio `pi1 / pi2`) can be specified. • *survival data*: a bound for testing H0: `hazard ratio = thetaH0 != 1` can be specified. 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`.
`pi1`	A numeric value or vector that represents 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)` (power calculations and simulations) or `seq(0.4, 0.6, 0.1)` (sample size calculations).
`pi2`	A numeric value that represents 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` which means that larger values of the test statistics yield smaller p-values.
`maxNumberOfSubjects`	`maxNumberOfSubjects > 0` needs to be specified. For two treatment arms, it is the maximum number of subjects for both treatment arms.

`allocationRatioPlanned`
The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.

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. Note that the power calculation for rates is always based on the normal approximation.

Value

Returns a `TrialDesignPlan` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

Other power functions: `getPowerMeans()`, `getPowerSurvival()`

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 = 1,
  directionUpper = NA,
  pi1 = NA_real_,
  pi2 = NA_real_,
  lambda1 = NA_real_,
  lambda2 = NA_real_,
  median1 = NA_real_,
  median2 = NA_real_,
  kappa = 1,
  hazardRatio = NA_real_,
  piecewiseSurvivalTime = NA_real_,
  allocationRatioPlanned = 1,
  eventTime = 12,
  accrualTime = c(0, 12),
  accrualIntensity = 0.1,
  accrualIntensityType = c("auto", "absolute", "relative"),
  maxNumberOfSubjects = NA_real_,
  maxNumberOfEvents = NA_real_,
```

```
  dropoutRate1 = 0,
  dropoutRate2 = 0,
  dropoutTime = 12
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to 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 Schoenfeld's formula can be used.
`thetaH0`	The null hypothesis value, default is `0` for the normal and the binary case (testing means and rates, respectively), it is `1` for the survival case (testing the hazard ratio). For non-inferiority designs, `thetaH0` is the non-inferiority bound. That is, in case of (one-sided) testing of • *means*: a value `!= 0` (or a value `!= 1` for testing the mean ratio) can be specified. • *rates*: a value `!= 0` (or a value `!= 1` for testing the risk ratio `pi1 / pi2`) can be specified. • *survival data*: a bound for testing H0: `hazard ratio = thetaH0 != 1` can be specified. 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`.
`directionUpper`	Specifies the direction of the alternative, only applicable for one-sided testing; default is `TRUE` which means that larger values of the test statistics yield smaller p-values.
`pi1`	A numeric value or vector that represents the assumed event rate in the treatment group, default is `seq(0.2, 0.5, 0.1)` (power calculations and simulations) or `seq(0.4, 0.6, 0.1)` (sample size calculations).
`pi2`	A numeric value that represents 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).

`median1`	The assumed median survival time in the treatment group, there is no default.
`median2`	The assumed median survival time in the reference group, there is no default.
`kappa`	A numeric value > 0. A `kappa != 1` will be used for the specification of the shape of the Weibull distribution. Default is `1`, i.e., the exponential survival distribution is used instead of the Weibull distribution. Note that the Weibull distribution cannot be used for the piecewise definition of the survival time distribution, i.e., only `piecewiselambda` (as a single value) and `kappa` can be specified. This function is equivalent to `pweibull(t, shape = kappa, scale = 1 / lambda)` of the `stats` package, 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.
`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, there is no default.
`piecewiseSurvivalTime`	A vector that specifies the time intervals for the piecewise definition of the exponential survival time cumulative distribution function (for details see `getPiecewiseSurvivalTime`).
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`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)` (for details see `getAccrualTime`).
`accrualIntensity`	A vector of accrual intensities, default is the relative intensity `0.1` (for details see `getAccrualTime`).
`accrualIntensityType`	A character value specifying the accrual intensity input type. Must be one of `"auto"`, `"absolute"`, or `"relative"`; default is `"auto"`, i.e., if all values are < 1 the type is `"relative"`, otherwise it is `"absolute"`.
`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, it 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 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.

Value

Returns a `TrialDesignPlan` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

Piecewise survival time

The first element of the vector `piecewiseSurvivalTime` 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).

Staggered patient entry

`accrualTime` is the time period of subjects' accrual in a study. It can be a value that defines the end of accrual or a vector. In this case, `accrualTime` can be used to define a non-constant accrual over time. For this, `accrualTime` is a vector that defines the accrual intervals. The first element of `accrualTime` must be equal to `0` and, additionally, `accrualIntensity` needs to be specified. `accrualIntensity` itself is a value or a vector (depending on the length of `accrualtime`) that defines the intensity how subjects enter the trial in the intervals defined through `accrualTime`.

`accrualTime` can also be a list that combines the definition of the accrual time and accrual intensity (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), `maxNumberOfSubjects > 0` needs to be specified and the end of accrual is calculated. In that case, `accrualIntensity` is the number of subjects per time unit, i.e., the absolute accrual intensity.

If the length of `accrualTime` equals the length of `accrualIntensity - 1` (i.e., the end of accrual is defined), `maxNumberOfSubjects` is calculated if the absolute accrual intensity is given. If all elements in `accrualIntensity` are smaller than 1, `accrualIntensity` defines the *relative* intensity how subjects enter the trial. For example, `accrualIntensity = c(0.1, 0.2)` specifies that in the second accrual interval the intensity is doubled as compared to the first accrual interval. The actual (absolute) accrual intensity is calculated for the calculated or given `maxNumberOfSubjects`. Note that the default is `accrualIntensity = 0.1` meaning that the *absolute* accrual intensity will be calculated.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

Other power functions: `getPowerMeans()`, `getPowerRates()`

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(20, 30))

# 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 = FALSE)

# 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 = 0.3, eventTime = 24, maxNumberOfEvents = 40, maxNumberOfSubjects = 200,
    directionUpper = FALSE)

# 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
getPowerSurvival(design = getDesignGroupSequential(kMax = 2), hazardRatio = 0.5,
    lambda2 = 0.02, maxNumberOfEvents = 40, maxNumberOfSubjects = 200,
    directionUpper = FALSE)

# 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
getPowerSurvival(median1 = 5, median2 = 3,
    maxNumberOfEvents = 40, maxNumberOfSubjects = 200, directionUpper = FALSE)

# Specify effect size based on median survival times of
# Weibull distribtion with kappa = 2
getPowerSurvival(median1 = 5, median2 = 3, kappa = 2,
    maxNumberOfEvents = 40, maxNumberOfSubjects = 200, directionUpper = FALSE)
```

`getRawData` *Get Simulation Raw Data for Survival*

Description

Returns the raw survival data which was generated for simulation.

Usage

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

Arguments

`x`	A `SimulationResults` object created by `getSimulationSurvival`.
`aggregate`	Logical. 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 `maxNumberOfRawDatasetsPerStage` > 0 (default is `0`).

This function can be used to get the simulated raw data from a simulation results object obtained by `getSimulationSurvival`. 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.

Value

Returns a `data.frame`.

Examples

```
results <- getSimulationSurvival(pi1 = seq(0.3,0.6,0.1), pi2 = 0.3, eventTime = 12,
    accrualTime = 24, plannedEvents = 40, maxNumberOfSubjects = 200,
    maxNumberOfIterations = 50, maxNumberOfRawDatasetsPerStage = 5)
rawData <- getRawData(results)
head(rawData)
dim(rawData)
```

`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,
  ...,
  directionUpper = TRUE,
  tolerance = 1e-06,
  stage = NA_integer_
)
```

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` and should be created with the function `getDataset`. For more information see `getDataset`.
`...`	Further arguments to be passed to methods (cf. separate functions in "See Also" below), e.g., `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 exact 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 `TRUE`. `intersectionTest` Defines the multiple test for the intersection hypotheses in the closed system of hypotheses when testing multiple hypotheses. Five options are available in multi-arm designs: `"Dunnett"`, `"Bonferroni"`, `"Simes"`, `"Sidak"`, and `"Hierarchical"`, default is `"Dunnett"`. Four options are available in population enrichment designs: `"SpiessensDebois"` (one subset only), `"Bonferroni"`, `"Simes"`, and `"Sidak"`, default is `"Simes"`. `varianceOption` Defines the way to calculate the variance in multiple treatment arms (> 2) or population enrichment designs for testing means. For multiple arms, three options are available: `"overallPooled"`, `"pairwisePooled"`, and `"notPooled"`, default is `"overallPooled"`. For enrichment designs, the options are: `"pooled"`, `"pooledFromFull"` (one subset only), and `"notPooled"`, default is `"pooled"`. `stratifiedAnalysis` For enrichment designs, typically a stratified analysis should be chosen. For testing means and rates, also a non-stratified analysis based on overall data can be performed. For survival data, only a stratified analysis is possible (see Brannath et al., 2009), default is `TRUE`.
`directionUpper`	Specifies the direction of the alternative, only applicable for one-sided testing; default is `TRUE` which means that larger values of the test statistics yield smaller p-values.
`tolerance`	The numerical tolerance, default is `1e-06`.
`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.

Value

Returns a `matrix` with 2 rows and `kMax` columns containing the lower RCI limits in the first row and the upper RCI limits in the second row, where each column represents a stage.

See Also

Other analysis functions: `getAnalysisResults()`, `getClosedCombinationTestResults()`, `getClosedConditionalDunn`
`getConditionalPower()`, `getConditionalRejectionProbabilities()`, `getFinalConfidenceInterval()`, `getFinalPValue()`, `getRepeatedPValues()`, `getStageResults()`, `getTestActions()`

Examples

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

`getRepeatedPValues` *Get Repeated P Values*

Description

Calculates the repeated p-values for a given test results.

Usage

```
getRepeatedPValues(stageResults, ..., tolerance = 1e-06)
```

Arguments

`stageResults`	The results at given stage, obtained from `getStageResults`.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`tolerance`	The numerical tolerance, default is `1e-06`.

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.

In multi-arm trials, the repeated p-values are defined separately for each treatment comparison within the closed testing procedure.

Value

Returns a `numeric` vector of length `kMax` or in case of multi-arm stage results a `matrix` (each column represents a stage, each row a comparison) containing the repeated p values.

Note on the dependency of `mnormt`

If `intersectionTest = "Dunnett"` or `intersectionTest = "SpiessensDebois"`, or the design is a conditional Dunnett design and the dataset is a multi-arm or enrichment dataset, `rpact` uses the R package mnormt to calculate the analysis results.

See Also

Other analysis functions: `getAnalysisResults()`, `getClosedCombinationTestResults()`, `getClosedConditionalDunn` `getConditionalPower()`, `getConditionalRejectionProbabilities()`, `getFinalConfidenceInterval()`, `getFinalPValue()`, `getRepeatedConfidenceIntervals()`, `getStageResults()`, `getTestActions()`

Examples

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

`getSampleSizeMeans` *Get Sample Size Means*

Description

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

Usage

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

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to 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`.
`normalApproximation`	The type of computation of the p-values. If `TRUE`, the variance is assumed to be known, default is `FALSE`, i.e., the calculations are performed with the t distribution.
`meanRatio`	If `TRUE`, the sample size for one-sided testing of H0: `mu1 / mu2 = thetaH0` is calculated, default is `FALSE`.
`thetaH0`	The null hypothesis value, default is `0` for the normal and the binary case (testing means and rates, respectively), it is `1` for the survival case (testing the hazard ratio). For non-inferiority designs, `thetaH0` is the non-inferiority bound. That is, in case of (one-sided) testing of • *means*: a value `!= 0` (or a value `!= 1` for testing the mean ratio) can be specified. • *rates*: a value `!= 0` (or a value `!= 1` for testing the risk ratio `pi1 / pi2`) can be specified. • *survival data*: a bound for testing H0: `hazard ratio = thetaH0 != 1` can be specified. 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`.
`alternative`	The alternative hypothesis value for testing means. This can be a vector of assumed alternatives, default is `seq(0, 1, 0.2)` (power calculations) or `seq(0.2, 1, 0.2)` (sample size calculations).
`stDev`	The standard deviation under which the sample size or power calculation is performed, default is `1`. If `meanRatio = TRUE` is specified, `stDev` defines the coefficient of variation `sigma / mu2`.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` 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 `TrialDesignPlan` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

Other sample size functions: `getSampleSizeRates()`, `getSampleSizeSurvival()`

Examples

```
# Calculate sample sizes in a fixed sample size parallel group design
# with allocation ratio \code{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,
  ...,
  groups = 2,
  normalApproximation = TRUE,
  riskRatio = FALSE,
  thetaH0 = ifelse(riskRatio, 1, 0),
  pi1 = c(0.4, 0.5, 0.6),
  pi2 = 0.2,
  allocationRatioPlanned = NA_real_
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to 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`.
`normalApproximation`	If `FALSE`, the sample size for the case of one treatment group is calculated exactly using the binomial distribution, default is `TRUE`.
`riskRatio`	If `TRUE`, the sample size for one-sided testing of H0: `pi1 / pi2 = thetaH0` is calculated, default is `FALSE`.
`thetaH0`	The null hypothesis value, default is `0` for the normal and the binary case (testing means and rates, respectively), it is `1` for the survival case (testing the hazard ratio). For non-inferiority designs, `thetaH0` is the non-inferiority bound. That is, in case of (one-sided) testing of • *means*: a value `!= 0` (or a value `!= 1` for testing the mean ratio) can be specified. • *rates*: a value `!= 0` (or a value `!= 1` for testing the risk ratio `pi1 / pi2`) can be specified.

- *survival data*: a bound for testing H0: `hazard ratio = thetaH0 != 1` can be specified.

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`.

`pi1`	A numeric value or vector that represents 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)` (power calculations and simulations) or `seq(0.4, 0.6, 0.1)` (sample size calculations).
`pi2`	A numeric value that represents the assumed probability in the reference group if two treatment groups are considered, default is `0.2`.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` 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 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 `TrialDesignPlan` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

Other sample size functions: `getSampleSizeMeans()`, `getSampleSizeSurvival()`

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(getDesignGroupSequential(kMax = 2, alpha = 0.05,
    beta = 0.1), 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), 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 = 1,
  pi1 = NA_real_,
  pi2 = NA_real_,
  lambda1 = NA_real_,
  lambda2 = NA_real_,
  median1 = NA_real_,
  median2 = NA_real_,
  kappa = 1,
  hazardRatio = NA_real_,
  piecewiseSurvivalTime = NA_real_,
  allocationRatioPlanned = NA_real_,
  eventTime = 12,
```

```
  accrualTime = c(0, 12),
  accrualIntensity = 0.1,
  accrualIntensityType = c("auto", "absolute", "relative"),
  followUpTime = NA_real_,
  maxNumberOfSubjects = NA_real_,
  dropoutRate1 = 0,
  dropoutRate2 = 0,
  dropoutTime = 12
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to 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 Schoenfeld's formula can be used.
`thetaH0`	The null hypothesis value, default is `0` for the normal and the binary case (testing means and rates, respectively), it is `1` for the survival case (testing the hazard ratio). For non-inferiority designs, `thetaH0` is the non-inferiority bound. That is, in case of (one-sided) testing of • *means*: a value `!= 0` (or a value `!= 1` for testing the mean ratio) can be specified. • *rates*: a value `!= 0` (or a value `!= 1` for testing the risk ratio `pi1 / pi2`) can be specified. • *survival data*: a bound for testing H0: `hazard ratio = thetaH0 != 1` can be specified. 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`.
`pi1`	A numeric value or vector that represents the assumed event rate in the treatment group, default is `seq(0.2, 0.5, 0.1)` (power calculations and simulations) or `seq(0.4, 0.6, 0.1)` (sample size calculations).
`pi2`	A numeric value that represents 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).

`median1`	The assumed median survival time in the treatment group, there is no default.
`median2`	The assumed median survival time in the reference group, there is no default.
`kappa`	A numeric value > 0. A `kappa != 1` will be used for the specification of the shape of the Weibull distribution. Default is `1`, i.e., the exponential survival distribution is used instead of the Weibull distribution. Note that the Weibull distribution cannot be used for the piecewise definition of the survival time distribution, i.e., only `piecewiselambda` (as a single value) and `kappa` can be specified. This function is equivalent to `pweibull(t, shape = kappa, scale = 1 / lambda)` of the `stats` package, 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.
`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, there is no default.
`piecewiseSurvivalTime`	A vector that specifies the time intervals for the piecewise definition of the exponential survival time cumulative distribution function (for details see `getPiecewiseSurvivalTime`).
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` 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.
`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)` (for details see `getAccrualTime`).
`accrualIntensity`	A vector of accrual intensities, default is the relative intensity `0.1` (for details see `getAccrualTime`).
`accrualIntensityType`	A character value specifying the accrual intensity input type. Must be one of `"auto"`, `"absolute"`, or `"relative"`; default is `"auto"`, i.e., if all values are < 1 the type is `"relative"`, otherwise it is `"absolute"`.
`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.

Optional argument `accountForObservationTimes`: if `accountForObservationTimes` = `TRUE`, the number of subjects is calculated assuming specific accrual and follow-up time, default is `TRUE`.

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.

Optional argument `accountForObservationTimes`: if `accountForObservationTimes` = `FALSE`, only the event rates are used for the calculation of the maximum number of subjects.

Value

Returns a `TrialDesignPlan` object. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

Piecewise survival time

The first element of the vector `piecewiseSurvivalTime` 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).

Staggered patient entry

`accrualTime` is the time period of subjects' accrual in a study. It can be a value that defines the end of accrual or a vector. In this case, `accrualTime` can be used to define a non-constant accrual over time. For this, `accrualTime` is a vector that defines the accrual intervals. The first element of `accrualTime` must be equal to `0` and, additionally, `accrualIntensity` needs to be specified. `accrualIntensity` itself is a value or a vector (depending on the length of `accrualtime`) that defines the intensity how subjects enter the trial in the intervals defined through `accrualTime`.

`accrualTime` can also be a list that combines the definition of the accrual time and accrual intensity (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), `maxNumberOfSubjects > 0` needs to be specified and the end of accrual is calculated. In that case, `accrualIntensity` is the number of subjects per time unit, i.e., the absolute accrual intensity.

If the length of `accrualTime` equals the length of `accrualIntensity - 1` (i.e., the end of accrual is defined), `maxNumberOfSubjects` is calculated if the absolute accrual intensity is given. If all elements in `accrualIntensity` are smaller than 1, `accrualIntensity` defines the *relative* intensity how subjects enter the trial. For example, `accrualIntensity = c(0.1, 0.2)` specifies that in the second accrual interval the intensity is doubled as compared to the first accrual interval. The actual (absolute) accrual intensity is calculated for the calculated or given `maxNumberOfSubjects`. Note that the default is `accrualIntensity = 0.1` meaning that the *absolute* accrual intensity will be calculated.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

Other sample size functions: `getSampleSizeMeans()`, `getSampleSizeRates()`

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 months.
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
getSampleSizeSurvival(median1 = 5, median2 = 3)

# Specify effect size based on median survival times of Weibull distribtion with
# kappa = 2
getSampleSizeSurvival(median1 = 5, median2 = 3, kappa = 2)

# 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)
```

`getSimulationEnrichmentMeans`
Get Simulation Enrichment Means

Description

Returns the simulated power, stopping and selection probabilities, conditional power, and expected sample size or testing means in an enrichment design testing situation.

Usage

```
getSimulationEnrichmentMeans(
  design = NULL,
  ...,
  effectList = NULL,
  intersectionTest = c("Simes", "SpiessensDebois", "Bonferroni", "Sidak"),
  stratifiedAnalysis = TRUE,
  adaptations = NA,
  typeOfSelection = c("best", "rBest", "epsilon", "all", "userDefined"),
  effectMeasure = c("effectEstimate", "testStatistic"),
  successCriterion = c("all", "atLeastOne"),
  epsilonValue = NA_real_,
  rValue = NA_real_,
  threshold = -Inf,
```

```
  plannedSubjects = NA_integer_,
  allocationRatioPlanned = NA_real_,
  minNumberOfSubjectsPerStage = NA_real_,
  maxNumberOfSubjectsPerStage = NA_real_,
  conditionalPower = NA_real_,
  thetaH1 = NA_real_,
  stDevH1 = NA_real_,
  maxNumberOfIterations = 1000L,
  seed = NA_real_,
  calcSubjectsFunction = NULL,
  selectPopulationsFunction = NULL,
  showStatistics = FALSE
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`effectList`	List of subsets, prevalences, and effect sizes with columns and number of rows reflecting the different situations to consider (see examples).
`intersectionTest`	Defines the multiple test for the intersection hypotheses in the closed system of hypotheses. Four options are available in enrichment designs: `"SpiessensDebois"`, `"Bonferroni"`, `"Simes"`, and `"Sidak"`, default is `"Simes"`.
`stratifiedAnalysis`	For enrichment designs, typically a stratified analysis should be chosen. For testing rates, also a non-stratified analysis based on overall data can be performed. For survival data, only a stratified analysis is possible (see Brannath et al., 2009), default is `TRUE`.
`adaptations`	A vector of length `kMax - 1` indicating whether or not an adaptation takes place at interim k, default is `rep(TRUE, kMax - 1)`.
`typeOfSelection`	The way the treatment arms or populations are selected at interim. Five options are available: `"best"`, `"rbest"`, `"epsilon"`, `"all"`, and `"userDefined"`, default is `"best"`. For `"rbest"` (select the `rValue` best treatment arms/populations), the parameter `rValue` has to be specified, for `"epsilon"` (select treatment arm/population not worse than epsilon compared to the best), the parameter `epsilonValue` has to be specified. If `"userDefined"` is selected, `"selectArmsFunction"` or `"selectPopulationsFunction"` has to be specified.
`effectMeasure`	Criterion for treatment arm/population selection, either based on test statistic (`"testStatistic"`) or effect estimate (difference for means and rates or ratio for survival) (`"effectEstimate"`), default is `"effectEstimate"`.

`successCriterion`	Defines when the study is stopped for efficacy at interim. Two options are available: `"all"` stops the trial if the efficacy criterion is fulfilled for all selected treatment arms/populations, `"atLeastOne"` stops if at least one of the selected treatment arms/populations is shown to be superior to control at interim, default is `"all"`.
`epsilonValue`	For `typeOfSelection = "epsilon"` (select treatment arm / population not worse than epsilon compared to the best), the parameter `epsilonValue` has to be specified.
`rValue`	For `typeOfSelection = "rbest"` (select the `rValue` best treatment arms / populations), the parameter `rValue` has to be specified.
`threshold`	Selection criterion: treatment arm / population is selected only if `effectMeasure` exceeds `threshold`, default is `-Inf`. `threshold` can also be a vector of length `activeArms` referring to a separate threshold condition over the treatment arms.
`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. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs, `plannedSubjects` refers to the number of subjects per selected active arm.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`minNumberOfSubjectsPerStage`	When performing a data driven sample size recalculation, the vector `minNumberOfSubjectsPerStage` with length kMax determines the minimum number of subjects per stage (i.e., not cumulated), the first element is not taken into account. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs `minNumberOfSubjectsPerStage` refers to the minimum number of subjects per selected active arm.
`maxNumberOfSubjectsPerStage`	When performing a data driven sample size recalculation, the vector `maxNumberOfSubjectsPerStage` with length kMax determines the maximum number of subjects per stage (i.e., not cumulated), the first element is not taken into account. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs `maxNumberOfSubjectsPerStage` refers to the maximum number of subjects per selected active arm.
`conditionalPower`	If `conditionalPower` together with `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerSt` (or `minNumberOfEventsPerStage` and `maxNumberOfEventsPerStage` for survival designs) is specified, a sample size recalculation based on the specified conditional power is performed. It is defined as the power for the subsequent stage given the current data. By default, the conditional power will be calculated under the observed effect size. Optionally, you can also specify `thetaH1` and `stDevH1` (for simulating means), `pi1H1` and `pi2H1` (for simulating rates), or `thetaH1` (for simulating hazard ratios) as parameters under which it is calculated and the sample size recalculation is performed.

`thetaH1`	If specified, the value of the alternative under which the conditional power or sample size recalculation calculation is performed.
`stDevH1`	If specified, the value of the standard deviation under which the conditional power or sample size recalculation calculation is performed, default is the value of `stDev`.
`maxNumberOfIterations`	The number of simulation iterations, default is `1000`.
`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 recalculation is performed with conditional power with specified `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerStage` (see details and examples).
`selectPopulationsFunction`	Optionally, a function can be entered that defines the way of how populations are selected. This function is allowed to depend on `effectVector` with length `populations` and `stage` (see examples).
`showStatistics`	If `TRUE`, summary statistics of the simulated data are displayed for the `print` command, otherwise the output is suppressed, default is `FALSE`.

Details

At given design the function simulates the power, stopping probabilities, selection probabilities, and expected sample size at given number of subjects, parameter configuration, and population selection rule in the enrichment situation. An allocation ratio can be specified referring to the ratio of number of subjects in the active treatment groups as compared to the control group.

The definition of `thetaH1` and/or `stDevH1` makes only sense if `kMax` > 1 and if `conditionalPower`, `minNumberOfSubjectsPerStage`, and `maxNumberOfSubjectsPerStage` (or `calcSubjectsFunction`) are defined.

`calcSubjectsFunction`
This function returns the number of subjects at given conditional power and conditional critical value for specified testing situation. The function might depend on the variables `stage`, `selectedPopulations`, `plannedSubjects`, `allocationRatioPlanned`, `minNumberOfSubjectsPerStage`, `maxNumberOfSubjectsPerStage`, `conditionalPower`, `conditionalCriticalValue`, `overallEffects`, and `stDevH1`. The function has to contain the three-dots argument '...' (see examples).

Value

Returns a `SimulationResults` object. The following generics (R generic functions) are available for this object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

Examples

```
# Assess a population selection strategy with one subset population.
# If the subset is better than the full population, then the subset
# is selected for the second stage, otherwise the full. Print and plot
# design characteristics.

# Define design
ds <- getDesignInverseNormal(kMax = 2)

# Define subgroups and their prevalences
subGroups <- c("S", "R") # fixed names!
prevalences <- c(0.2, 0.8)

# Define effect matrix and variability
effectR <- 0.2
m <- c()
for (effectS in seq(0, 0.5, 0.25)) {
    m <- c(m, effectS, effectR)
}
effects <- matrix(m, byrow = TRUE, ncol = 2)
stDev <- c(0.4, 0.8)

# Define effect list
el <- list(subGroups=subGroups, prevalences=prevalences, stDevs = stDev, effects = effects)

# Perform simulation
simResultsPE <- getSimulationEnrichmentMeans(design = ds,
    effectList = el, plannedSubjects = c(50, 100),
    maxNumberOfIterations = 100)
print(simResultsPE)

# Assess the design characteristics of a user defined selection
# strategy in a three-stage design with no interim efficacy stop
# using the inverse normal method for combining the stages.
# Only the second interim is used for a selecting of a study
# population. There is a small probability for stopping the trial
# at the first interim.

# Define design
ds <- getDesignInverseNormal(typeOfDesign = "noEarlyEfficacy", kMax = 3)

# Define selection function
mySelection <- function(effectVector, stage) {
```

```
    selectedPopulations <- rep(TRUE, 3)
    if (stage == 2) {
        selectedPopulations <- (effectVector >= c(1, 2, 3))
    }
    return(selectedPopulations)
}

# Define subgroups and their prevalences
subGroups <- c("S1", "S12", "S2", "R")   # fixed names!
prevalences <- c(0.2, 0.3, 0.4, 0.1)

effectR <- 1.5
effectS12 = 5
m <- c()
for (effectS1 in seq(0, 5, 1)) {
    for (effectS2 in seq(0, 5, 1)) {
        m <- c(m, effectS1, effectS12, effectS2, effectR)
    }
}
effects <- matrix(m, byrow = TRUE, ncol = 4)
stDev <- 10

# Define effect list
el <- list(subGroups=subGroups, prevalences=prevalences, stDevs = stDev, effects = effects)

# Perform simulation
simResultsPE <- getSimulationEnrichmentMeans(design = ds,
    effectList = el,
    typeOfSelection = "userDefined",
    selectPopulationsFunction = mySelection,
    intersectionTest = "Simes",
    plannedSubjects = c(50, 100, 150),
    maxNumberOfIterations = 100)
print(simResultsPE)
if (require(ggplot2)) plot(simResultsPE, type = 3)
```

`getSimulationEnrichmentRates`

Get Simulation Enrichment Rates

Description

Returns the simulated power, stopping and selection probabilities, conditional power, and expected sample size for testing rates in an enrichment design testing situation.

Usage

```
getSimulationEnrichmentRates(
```

```
  design = NULL,
  ...,
  effectList = NULL,
  intersectionTest = c("Simes", "SpiessensDebois", "Bonferroni", "Sidak"),
  stratifiedAnalysis = TRUE,
  directionUpper = TRUE,
  adaptations = NA,
  typeOfSelection = c("best", "rBest", "epsilon", "all", "userDefined"),
  effectMeasure = c("effectEstimate", "testStatistic"),
  successCriterion = c("all", "atLeastOne"),
  epsilonValue = NA_real_,
  rValue = NA_real_,
  threshold = -Inf,
  plannedSubjects = NA_real_,
  allocationRatioPlanned = NA_real_,
  minNumberOfSubjectsPerStage = NA_real_,
  maxNumberOfSubjectsPerStage = NA_real_,
  conditionalPower = NA_real_,
  piTreatmentH1 = NA_real_,
  piControlH1 = NA_real_,
  maxNumberOfIterations = 1000L,
  seed = NA_real_,
  calcSubjectsFunction = NULL,
  selectPopulationsFunction = NULL,
  showStatistics = FALSE
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`effectList`	List of subsets, prevalences, and effect sizes with columns and number of rows reflecting the different situations to consider (see examples).
`intersectionTest`	Defines the multiple test for the intersection hypotheses in the closed system of hypotheses. Four options are available in enrichment designs: `"SpiessensDebois"`, `"Bonferroni"`, `"Simes"`, and `"Sidak"`, default is `"Simes"`.
`stratifiedAnalysis`	For enrichment designs, typically a stratified analysis should be chosen. For testing rates, also a non-stratified analysis based on overall data can be performed. For survival data, only a stratified analysis is possible (see Brannath et al., 2009), default is `TRUE`.
`directionUpper`	Specifies the direction of the alternative, only applicable for one-sided testing; default is `TRUE` which means that larger values of the test statistics yield smaller p-values.

`adaptations`	A vector of length `kMax - 1` indicating whether or not an adaptation takes place at interim k, default is `rep(TRUE, kMax - 1)`.
`typeOfSelection`	The way the treatment arms or populations are selected at interim. Five options are available: `"best"`, `"rbest"`, `"epsilon"`, `"all"`, and `"userDefined"`, default is `"best"`. For `"rbest"` (select the `rValue` best treatment arms/populations), the parameter `rValue` has to be specified, for `"epsilon"` (select treatment arm/population not worse than epsilon compared to the best), the parameter `epsilonValue` has to be specified. If `"userDefined"` is selected, `"selectArmsFunction"` or `"selectPopulationsFunction"` has to be specified.
`effectMeasure`	Criterion for treatment arm/population selection, either based on test statistic (`"testStatistic"`) or effect estimate (difference for means and rates or ratio for survival) (`"effectEstimate"`), default is `"effectEstimate"`.
`successCriterion`	Defines when the study is stopped for efficacy at interim. Two options are available: `"all"` stops the trial if the efficacy criterion is fulfilled for all selected treatment arms/populations, `"atLeastOne"` stops if at least one of the selected treatment arms/populations is shown to be superior to control at interim, default is `"all"`.
`epsilonValue`	For `typeOfSelection = "epsilon"` (select treatment arm / population not worse than epsilon compared to the best), the parameter `epsilonValue` has to be specified.
`rValue`	For `typeOfSelection = "rbest"` (select the `rValue` best treatment arms / populations), the parameter `rValue` has to be specified.
`threshold`	Selection criterion: treatment arm / population is selected only if `effectMeasure` exceeds `threshold`, default is `-Inf`. `threshold` can also be a vector of length `activeArms` referring to a separate threshold condition over the treatment arms.
`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. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs, `plannedSubjects` refers to the number of subjects per selected active arm.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`minNumberOfSubjectsPerStage`	When performing a data driven sample size recalculation, the vector `minNumberOfSubjectsPerStage` with length kMax determines the minimum number of subjects per stage (i.e., not cumulated), the first element is not taken into account. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs `minNumberOfSubjectsPerStage` refers to the minimum number of subjects per selected active arm.

`maxNumberOfSubjectsPerStage`	When performing a data driven sample size recalculation, the vector `maxNumberOfSubjectsPerStage` with length kMax determines the maximum number of subjects per stage (i.e., not cumulated), the first element is not taken into account. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs `maxNumberOfSubjectsPerStage` refers to the maximum number of subjects per selected active arm.
`conditionalPower`	If `conditionalPower` together with `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerSt` (or `minNumberOfEventsPerStage` and `maxNumberOfEventsPerStage` for survival designs) is specified, a sample size recalculation based on the specified conditional power is performed. It is defined as the power for the subsequent stage given the current data. By default, the conditional power will be calculated under the observed effect size. Optionally, you can also specify `thetaH1` and `stDevH1` (for simulating means), `pi1H1` and `pi2H1` (for simulating rates), or `thetaH1` (for simulating hazard ratios) as parameters under which it is calculated and the sample size recalculation is performed.
`piTreatmentH1`	If specified, the assumed probabilities in the active arm under which the sample size recalculation was performed and the conditional power was calculated.
`piControlH1`	If specified, the assumed probabilities in the control arm under which the sample size recalculation was performed and the conditional power was calculated.
`maxNumberOfIterations`	The number of simulation iterations, default is `1000`.
`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 recalculation is performed with conditional power with specified `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerStage` (see details and examples).
`selectPopulationsFunction`	Optionally, a function can be entered that defines the way of how populations are selected. This function is allowed to depend on `effectVector` with length `populations` and `stage` (see examples).
`showStatistics`	If `TRUE`, summary statistics of the simulated data are displayed for the `print` command, otherwise the output is suppressed, default is `FALSE`.

Details

At given design the function simulates the power, stopping probabilities, selection probabilities, and expected sample size at given number of subjects, parameter configuration, and treatment arm selection rule in the enrichment situation. An allocation ratio can be specified referring to the ratio of number of subjects in the active treatment groups as compared to the control group.

The definition of `piTreatmentH1` and/or `piControlH1` makes only sense if `kMax` > 1 and if `conditionalPower`, `minNumberOfSubjectsPerStage`, and `maxNumberOfSubjectsPerStage` (or `calcSubjectsFunction`) are defined.

`calcSubjectsFunction`
This function returns the number of subjects at given conditional power and conditional critical

value for specified testing situation. The function might depend on the variables `stage`, `selectedPopulations`, `directionUpper`, `plannedSubjects`, `allocationRatioPlanned`, `minNumberOfSubjectsPerStage`, `maxNumberOfSubjectsPerStage`, `conditionalPower`, `conditionalCriticalValue`, `overallRatesTreatment`, `overallRatesControl`, `piTreatmentH1`, and `piControlH1`. The function has to contain the three-dots argument '...' (see examples).

Value

Returns a `SimulationResults` object. The following generics (R generic functions) are available for this object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

Examples

```
# Assess a population selection strategy with two subset populations and
# a binary endpoint using a stratified analysis. No early efficacy stop,
# weighted inverse normal method with weight sqrt(0.4).
pi2 <- c(0.3, 0.4, 0.3, 0.55)
pi1Seq <- seq(0.0, 0.2, 0.2)
pi1 <- matrix(rep(pi1Seq, length(pi2)), ncol = length(pi1Seq), byrow = TRUE) + pi2
effectList <- list(
    subGroups = c("S1", "S2", "S12", "R"),
    prevalences = c(0.1, 0.4, 0.2, 0.3),
    piControl = pi2,
    piTreatments = expand.grid(pi1[1, ], pi1[2, ], pi1[3, ], pi1[4, ])
)
ds <- getDesignInverseNormal(informationRates = c(0.4, 1),
    typeOfDesign = "noEarlyEfficacy")
simResultsPE <- getSimulationEnrichmentRates(ds, plannedSubjects = c(150, 300),
    allocationRatioPlanned = 1.5, directionUpper = TRUE,
    effectList = effectList, stratifiedAnalysis = TRUE,
    intersectionTest = "Sidak",
    typeOfSelection = "epsilon", epsilonValue = 0.025,
    maxNumberOfIterations = 100)
print(simResultsPE)
```

`getSimulationEnrichmentSurvival` *Get Simulation Enrichment Survival*

Description

Returns the simulated power, stopping and selection probabilities, conditional power, and expected sample size for testing hazard ratios in an enrichment design testing situation. In contrast to `getSimulationSurvival()` (where survival times are simulated), normally distributed logrank test statistics are simulated.

Usage

```
getSimulationEnrichmentSurvival(
  design = NULL,
  ...,
  effectList = NULL,
  intersectionTest = c("Simes", "SpiessensDebois", "Bonferroni", "Sidak"),
  stratifiedAnalysis = TRUE,
  directionUpper = TRUE,
  adaptations = NA,
  typeOfSelection = c("best", "rBest", "epsilon", "all", "userDefined"),
  effectMeasure = c("effectEstimate", "testStatistic"),
  successCriterion = c("all", "atLeastOne"),
  epsilonValue = NA_real_,
  rValue = NA_real_,
  threshold = -Inf,
  plannedEvents = NA_real_,
  allocationRatioPlanned = NA_real_,
  minNumberOfEventsPerStage = NA_real_,
  maxNumberOfEventsPerStage = NA_real_,
  conditionalPower = NA_real_,
  thetaH1 = NA_real_,
  maxNumberOfIterations = 1000L,
  seed = NA_real_,
  calcEventsFunction = NULL,
  selectPopulationsFunction = NULL,
  showStatistics = FALSE
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.

`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`effectList`	List of subsets, prevalences, and effect sizes with columns and number of rows reflecting the different situations to consider (see examples).
`intersectionTest`	Defines the multiple test for the intersection hypotheses in the closed system of hypotheses. Four options are available in enrichment designs: `"SpiessensDebois"`, `"Bonferroni"`, `"Simes"`, and `"Sidak"`, default is `"Simes"`.
`stratifiedAnalysis`	For enrichment designs, typically a stratified analysis should be chosen. For testing rates, also a non-stratified analysis based on overall data can be performed. For survival data, only a stratified analysis is possible (see Brannath et al., 2009), default is `TRUE`.
`directionUpper`	Specifies the direction of the alternative, only applicable for one-sided testing; default is `TRUE` which means that larger values of the test statistics yield smaller p-values.
`adaptations`	A vector of length `kMax - 1` indicating whether or not an adaptation takes place at interim k, default is `rep(TRUE, kMax - 1)`.
`typeOfSelection`	The way the treatment arms or populations are selected at interim. Five options are available: `"best"`, `"rbest"`, `"epsilon"`, `"all"`, and `"userDefined"`, default is `"best"`. For `"rbest"` (select the `rValue` best treatment arms/populations), the parameter `rValue` has to be specified, for `"epsilon"` (select treatment arm/population not worse than epsilon compared to the best), the parameter `epsilonValue` has to be specified. If `"userDefined"` is selected, `"selectArmsFunction"` or `"selectPopulationsFunction"` has to be specified.
`effectMeasure`	Criterion for treatment arm/population selection, either based on test statistic (`"testStatistic"`) or effect estimate (difference for means and rates or ratio for survival) (`"effectEstimate"`), default is `"effectEstimate"`.
`successCriterion`	Defines when the study is stopped for efficacy at interim. Two options are available: `"all"` stops the trial if the efficacy criterion is fulfilled for all selected treatment arms/populations, `"atLeastOne"` stops if at least one of the selected treatment arms/populations is shown to be superior to control at interim, default is `"all"`.
`epsilonValue`	For `typeOfSelection = "epsilon"` (select treatment arm / population not worse than epsilon compared to the best), the parameter `epsilonValue` has to be specified.
`rValue`	For `typeOfSelection = "rbest"` (select the `rValue` best treatment arms / populations), the parameter `rValue` has to be specified.
`threshold`	Selection criterion: treatment arm / population is selected only if `effectMeasure` exceeds `threshold`, default is `-Inf`. `threshold` can also be a vector of length `activeArms` referring to a separate threshold condition over the treatment arms.
`plannedEvents`	`plannedEvents` is a vector of length `kMax` (the number of stages of the design) that determines the number of cumulated (overall) events in survival designs

	when the interim stages are planned. For two treatment arms, it is the number of events for both treatment arms. For multi-arm designs, `plannedEvents` refers to the overall number of events for the selected arms plus control.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`minNumberOfEventsPerStage`	When performing a data driven sample size recalculation, the vector `minNumberOfEventsPerStage` with length kMax determines the minimum number of events per stage (i.e., not cumulated), the first element is not taken into account.
`maxNumberOfEventsPerStage`	When performing a data driven sample size recalculation, the vector `maxNumberOfEventsPerStage` with length kMax determines the maximum number of events per stage (i.e., not cumulated), the first element is not taken into account.
`conditionalPower`	If `conditionalPower` together with `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerSt` (or `minNumberOfEventsPerStage` and `maxNumberOfEventsPerStage` for survival designs) is specified, a sample size recalculation based on the specified conditional power is performed. It is defined as the power for the subsequent stage given the current data. By default, the conditional power will be calculated under the observed effect size. Optionally, you can also specify `thetaH1` and `stDevH1` (for simulating means), `pi1H1` and `pi2H1` (for simulating rates), or `thetaH1` (for simulating hazard ratios) as parameters under which it is calculated and the sample size recalculation is performed.
`thetaH1`	If specified, the value of the alternative under which the conditional power or sample size recalculation calculation is performed.
`maxNumberOfIterations`	The number of simulation iterations, default is `1000`.
`seed`	The seed to reproduce the simulation, default is a random seed.
`calcEventsFunction`	Optionally, a function can be entered that defines the way of performing the sample size recalculation. By default, sample size recalculation is performed with conditional power with specified `minNumberOfEventsPerStage` and `maxNumberOfEventsPerStage` (see details and examples).
`selectPopulationsFunction`	Optionally, a function can be entered that defines the way of how populations are selected. This function is allowed to depend on `effectVector` with length `populations` and `stage` (see examples).
`showStatistics`	If `TRUE`, summary statistics of the simulated data are displayed for the `print` command, otherwise the output is suppressed, default is `FALSE`.

Details

At given design the function simulates the power, stopping probabilities, selection probabilities, and expected event number at given number of events, parameter configuration, and population selection rule in the enrichment situation. An allocation ratio can be specified referring to the ratio of number of subjects in the active treatment group as compared to the control group.

The definition of `thetaH1` makes only sense if `kMax` > 1 and if `conditionalPower`, `minNumberOfEventsPerStage`, and `maxNumberOfEventsPerStage` (or `calcEventsFunction`) are defined.

`calcEventsFunction`

This function returns the number of events at given conditional power and conditional critical value for specified testing situation. The function might depend on the variables `stage`, `selectedPopulations`, `plannedEvents`, `directionUpper`, `allocationRatioPlanned`, `minNumberOfEventsPerStage`, `maxNumberOfEventsPerSt` `conditionalPower`, `conditionalCriticalValue`, and `overallEffects`. The function has to contain the three-dots argument '...' (see examples).

Value

Returns a `SimulationResults` object. The following generics (R generic functions) are available for this object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

Examples

```
# Assess a population selection strategy with one subset population and
# a survival endpoint. The considered situations are defined through the
# event rates yielding a range of hazard ratios in the subsets. Design
# with O'Brien and Fleming alpha spending and a reassessment of event
# number in the first interim based on conditional power and assumed
# hazard ratio using weighted inverse normal combination test.

subGroups <- c("S", "R")
prevalences <- c(0.40, 0.60)

p2 <- c(0.3, 0.4)
range1 <- p2[1] + seq(0, 0.3, 0.05)

p1 <- c()
for (x1 in range1) {
```

```
    p1 <- c(p1, x1, p2[2] + 0.1)
}
hazardRatios <- log(matrix(1 - p1, byrow = TRUE, ncol = 2)) /
    matrix(log(1 - p2), byrow = TRUE, ncol = 2,
    nrow = length(range1))

effectList <- list(subGroups=subGroups, prevalences=prevalences,
    hazardRatios = hazardRatios)

ds <- getDesignInverseNormal(informationRates = c(0.3, 0.7, 1),
typeOfDesign = "asOF")

simResultsPE <- getSimulationEnrichmentSurvival(ds,
    plannedEvents = c(40, 90, 120),
    effectList = effectList,
    typeOfSelection = "rbest", rValue = 2,
    conditionalPower = 0.8, minNumberOfEventsPerStage = c(NA, 50, 30),
    maxNumberOfEventsPerStage = c(NA, 150, 30), thetaH1 = 4/3,
    maxNumberOfIterations = 100)
print(simResultsPE)
```

`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,
  normalApproximation = TRUE,
  meanRatio = FALSE,
  thetaH0 = ifelse(meanRatio, 1, 0),
  alternative = seq(0, 1, 0.2),
  stDev = 1,
  plannedSubjects = NA_real_,
  directionUpper = TRUE,
  allocationRatioPlanned = NA_real_,
  minNumberOfSubjectsPerStage = NA_real_,
  maxNumberOfSubjectsPerStage = NA_real_,
  conditionalPower = NA_real_,
  thetaH1 = NA_real_,
```

```
  stDevH1 = NA_real_,
  maxNumberOfIterations = 1000L,
  seed = NA_real_,
  calcSubjectsFunction = NULL,
  showStatistics = FALSE
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to 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`.
`normalApproximation`	The type of computation of the p-values. Default is `TRUE`, i.e., normally distributed test statistics are generated. If `FALSE`, the t test is used for calculating the p-values, i.e., t distributed test statistics are generated.
`meanRatio`	If `TRUE`, the design characteristics for one-sided testing of H0: `mu1 / mu2 = thetaH0` are simulated, default is `FALSE`.
`thetaH0`	The null hypothesis value, default is `0` for the normal and the binary case (testing means and rates, respectively), it is `1` for the survival case (testing the hazard ratio). For non-inferiority designs, `thetaH0` is the non-inferiority bound. That is, in case of (one-sided) testing of • *means*: a value `!= 0` (or a value `!= 1` for testing the mean ratio) can be specified. • *rates*: a value `!= 0` (or a value `!= 1` for testing the risk ratio `pi1 / pi2`) can be specified. • *survival data*: a bound for testing H0: `hazard ratio = thetaH0 != 1` can be specified. 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`.
`alternative`	The alternative hypothesis value for testing means under which the data is simulated. This can be a vector of assumed alternatives, default is `seq(0, 1, 0.2)`.
`stDev`	The standard deviation under which the data is simulated, 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. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs, `plannedSubjects` refers to the number of subjects per selected active arm.

`directionUpper`	Specifies the direction of the alternative, only applicable for one-sided testing; default is `TRUE` which means that larger values of the test statistics yield smaller p-values.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`minNumberOfSubjectsPerStage`	When performing a data driven sample size recalculation, the vector `minNumberOfSubjectsPerStage` with length kMax determines the minimum number of subjects per stage (i.e., not cumulated), the first element is not taken into account. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs `minNumberOfSubjectsPerStage` refers to the minimum number of subjects per selected active arm.
`maxNumberOfSubjectsPerStage`	When performing a data driven sample size recalculation, the vector `maxNumberOfSubjectsPerStage` with length kMax determines the maximum number of subjects per stage (i.e., not cumulated), the first element is not taken into account. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs `maxNumberOfSubjectsPerStage` refers to the maximum number of subjects per selected active arm.
`conditionalPower`	If `conditionalPower` together with `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerSt` (or `minNumberOfEventsPerStage` and `maxNumberOfEventsPerStage` for survival designs) is specified, a sample size recalculation based on the specified conditional power is performed. It is defined as the power for the subsequent stage given the current data. By default, the conditional power will be calculated under the observed effect size. Optionally, you can also specify `thetaH1` and `stDevH1` (for simulating means), `pi1H1` and `pi2H1` (for simulating rates), or `thetaH1` (for simulating hazard ratios) as parameters under which it is calculated and the sample size recalculation is performed.
`thetaH1`	If specified, the value of the alternative under which the conditional power or sample size recalculation calculation is performed.
`stDevH1`	If specified, the value of the standard deviation under which the conditional power or sample size recalculation calculation is performed, default is the value of `stDev`.
`maxNumberOfIterations`	The number of simulation iterations, default is `1000`.
`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 recalculation is performed with conditional power with specified `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerStage` (see details and examples).
`showStatistics`	If `TRUE`, summary statistics of the simulated data are displayed for the `print` command, otherwise the output is suppressed, default is `FALSE`.

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.

The definition of `thetaH1` makes only sense if `kMax` > 1 and if `conditionalPower`, `minNumberOfSubjectsPerStage`, and `maxNumberOfSubjectsPerStage` (or `calcSubjectsFunction`) are defined.

`calcSubjectsFunction`
This function returns the number of subjects at given conditional power and conditional critical value for specified testing situation. The function might depend on variables `stage`, `meanRatio`, `thetaH0`, `groups`, `plannedSubjects`, `sampleSizesPerStage`, `directionUpper`, `allocationRatioPlanned`, `minNumberOfSubjectsPerStage`, `maxNumberOfSubjectsPerStage`, `conditionalPower`, `conditionalCriticalValue`, `thetaH1`, and `stDevH1`. The function has to contain the three-dots argument '...' (see examples).

Value

Returns a `SimulationResults` object. The following generics (R generic functions) are available for this object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

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 Fisher's combination test).
8. `testStatisticsPerStage`: The test statistic for each stage if only data from the considered stage is taken into account.
9. `effectEstimate`: Overall simulated standardized 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`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

Examples

```
# Fixed sample size design with two groups, total sample size 40,
# alternative = c(0, 0.2, 0.4, 0.8, 1), and standard deviation = 1 (the default)
getSimulationMeans(plannedSubjects = 40, maxNumberOfIterations = 10)

# Increase number of simulation iterations and compare results
# with power calculator using normal approximation
getSimulationMeans(alternative = 0:4, stDev = 5,
    plannedSubjects = 40, maxNumberOfIterations = 1000)
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 minNumberOfSubjectsPerStage and
# maxNumberOfSubjectsPerStage
getSimulationMeans(designIN, alternative = 0:4, stDev = 5,
    plannedSubjects = c(20, 40, 60),
    minNumberOfSubjectsPerStage = c(NA, 20, 20),
    maxNumberOfSubjectsPerStage = c(NA, 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,
        minNumberOfSubjectsPerStage,
        maxNumberOfSubjectsPerStage,
        sampleSizesPerStage,
        conditionalPower,
        conditionalCriticalValue,
        thetaH1) {
    if (stage == 2) {
        stageSubjects <- 4 * (max(0, conditionalCriticalValue +
            stats::qnorm(conditionalPower)))^2 / (max(1e-12, thetaH1))^2
        stageSubjects <- min(max(minNumberOfSubjectsPerStage[stage],
            stageSubjects), maxNumberOfSubjectsPerStage[stage])
    } else {
        stageSubjects <- sampleSizesPerStage[stage - 1]
    }
    return(stageSubjects)
}
getSimulationMeans(designIN, alternative = 2:4, stDev = 5,
    plannedSubjects = c(20, 40, 60),
    minNumberOfSubjectsPerStage = c(NA, 20, 20),
    maxNumberOfSubjectsPerStage = c(NA, 160, 160),
    conditionalPower = 0.8,
    calcSubjectsFunction = mySampleSizeCalculationFunction,
    maxNumberOfIterations = 50)
```

`getSimulationMultiArmMeans` *Get Simulation Multi-Arm Means*

Description

Returns the simulated power, stopping and selection probabilities, conditional power, and expected sample size for testing means in a multi-arm treatment groups testing situation.

Usage

```
getSimulationMultiArmMeans(
  design = NULL,
  ...,
  activeArms = 3L,
  effectMatrix = NULL,
  typeOfShape = c("linear", "sigmoidEmax", "userDefined"),
  muMaxVector = seq(0, 1, 0.2),
  gED50 = NA_real_,
  slope = 1,
 intersectionTest = c("Dunnett", "Bonferroni", "Simes", "Sidak", "Hierarchical"),
  stDev = 1,
  adaptations = NA,
  typeOfSelection = c("best", "rBest", "epsilon", "all", "userDefined"),
  effectMeasure = c("effectEstimate", "testStatistic"),
  successCriterion = c("all", "atLeastOne"),
  epsilonValue = NA_real_,
  rValue = NA_real_,
  threshold = -Inf,
  plannedSubjects = NA_integer_,
  allocationRatioPlanned = NA_real_,
  minNumberOfSubjectsPerStage = NA_real_,
  maxNumberOfSubjectsPerStage = NA_real_,
  conditionalPower = NA_real_,
  thetaH1 = NA_real_,
  stDevH1 = NA_real_,
  maxNumberOfIterations = 1000L,
  seed = NA_real_,
  calcSubjectsFunction = NULL,
  selectArmsFunction = NULL,
  showStatistics = FALSE
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`activeArms`	The number of active treatment arms to be compared with control, default is `3`.
`effectMatrix`	Matrix of effect sizes with `activeArms` columns and number of rows reflecting the different situations to consider.
`typeOfShape`	The shape of the dose-response relationship over the treatment groups. This can be either `"linear"`, `"sigmoidEmax"`, or `"userDefined"`. If `"sigmoidEmax"` is selected, `"gED50"` and `"slope"` has to be entered to specify the ED50 and the slope of the sigmoid Emax model. For `"linear"` and `"sigmoidEmax"`,

	`"muMaxVector"` specifies the range of effect sizes for the treatment group with highest response. If `"userDefined"` is selected, `"effectMatrix"` has to be entered.
`muMaxVector`	Range of effect sizes for the treatment group with highest response for `"linear"` and `"sigmoidEmax"` model, default is `seq(0, 1, 0.2)`.
`gED50`	If `typeOfShape = "sigmoidEmax"` is selected, `"gED50"` has to be entered to specify the ED50 of the sigmoid Emax model.
`slope`	If `typeOfShape = "sigmoidEmax"` is selected, `"slope"` can be entered to specify the slope of the sigmoid Emax model, default is 1.
`intersectionTest`	Defines the multiple test for the intersection hypotheses in the closed system of hypotheses. Five options are available in multi-arm designs: `"Dunnett"`, `"Bonferroni"`, `"Simes"`, `"Sidak"`, and `"Hierarchical"`, default is `"Dunnett"`.
`stDev`	The standard deviation under which the data is simulated, default is `1`. If `meanRatio = TRUE` is specified, `stDev` defines the coefficient of variation `sigma / mu2`.
`adaptations`	A vector of length `kMax - 1` indicating whether or not an adaptation takes place at interim k, default is `rep(TRUE, kMax - 1)`.
`typeOfSelection`	The way the treatment arms or populations are selected at interim. Five options are available: `"best"`, `"rbest"`, `"epsilon"`, `"all"`, and `"userDefined"`, default is `"best"`. For `"rbest"` (select the `rValue` best treatment arms/populations), the parameter `rValue` has to be specified, for `"epsilon"` (select treatment arm/population not worse than epsilon compared to the best), the parameter `epsilonValue` has to be specified. If `"userDefined"` is selected, `"selectArmsFunction"` or `"selectPopulationsFunction"` has to be specified.
`effectMeasure`	Criterion for treatment arm/population selection, either based on test statistic (`"testStatistic"`) or effect estimate (difference for means and rates or ratio for survival) (`"effectEstimate"`), default is `"effectEstimate"`.
`successCriterion`	Defines when the study is stopped for efficacy at interim. Two options are available: `"all"` stops the trial if the efficacy criterion is fulfilled for all selected treatment arms/populations, `"atLeastOne"` stops if at least one of the selected treatment arms/populations is shown to be superior to control at interim, default is `"all"`.
`epsilonValue`	For `typeOfSelection = "epsilon"` (select treatment arm / population not worse than epsilon compared to the best), the parameter `epsilonValue` has to be specified.
`rValue`	For `typeOfSelection = "rbest"` (select the `rValue` best treatment arms / populations), the parameter `rValue` has to be specified.
`threshold`	Selection criterion: treatment arm / population is selected only if `effectMeasure` exceeds `threshold`, default is `-Inf`. `threshold` can also be a vector of length `activeArms` referring to a separate threshold condition over the treatment arms.
`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. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs, `plannedSubjects` refers to the number of subjects per selected active arm.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`minNumberOfSubjectsPerStage`	When performing a data driven sample size recalculation, the vector `minNumberOfSubjectsPerStage` with length kMax determines the minimum number of subjects per stage (i.e., not cumulated), the first element is not taken into account. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs `minNumberOfSubjectsPerStage` refers to the minimum number of subjects per selected active arm.
`maxNumberOfSubjectsPerStage`	When performing a data driven sample size recalculation, the vector `maxNumberOfSubjectsPerStage` with length kMax determines the maximum number of subjects per stage (i.e., not cumulated), the first element is not taken into account. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs `maxNumberOfSubjectsPerStage` refers to the maximum number of subjects per selected active arm.
`conditionalPower`	If `conditionalPower` together with `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerSt` (or `minNumberOfEventsPerStage` and `maxNumberOfEventsPerStage` for survival designs) is specified, a sample size recalculation based on the specified conditional power is performed. It is defined as the power for the subsequent stage given the current data. By default, the conditional power will be calculated under the observed effect size. Optionally, you can also specify `thetaH1` and `stDevH1` (for simulating means), `pi1H1` and `pi2H1` (for simulating rates), or `thetaH1` (for simulating hazard ratios) as parameters under which it is calculated and the sample size recalculation is performed.
`thetaH1`	If specified, the value of the alternative under which the conditional power or sample size recalculation calculation is performed.
`stDevH1`	If specified, the value of the standard deviation under which the conditional power or sample size recalculation calculation is performed, default is the value of `stDev`.
`maxNumberOfIterations`	The number of simulation iterations, default is `1000`.
`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 recalculation is performed with conditional power with specified `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerStage` (see details and examples).
`selectArmsFunction`	Optionally, a function can be entered that defines the way of how treatment arms are selected. This function is allowed to depend on `effectVector` with length `activeArms` and `stage` (see examples).

`showStatistics`	If `TRUE`, summary statistics of the simulated data are displayed for the `print` command, otherwise the output is suppressed, default is `FALSE`.

Details

At given design the function simulates the power, stopping probabilities, selection probabilities, and expected sample size at given number of subjects, parameter configuration, and treatment arm selection rule in the multi-arm situation. An allocation ratio can be specified referring to the ratio of number of subjects in the active treatment groups as compared to the control group.

The definition of `thetaH1` and/or `stDevH1` makes only sense if `kMax` > 1 and if `conditionalPower`, `minNumberOfSubjectsPerStage`, and `maxNumberOfSubjectsPerStage` (or `calcSubjectsFunction`) are defined.

`calcSubjectsFunction`
This function returns the number of subjects at given conditional power and conditional critical value for specified testing situation. The function might depend on the variables `stage`, `selectedArms`, `plannedSubjects`, `allocationRatioPlanned`, `minNumberOfSubjectsPerStage`, `maxNumberOfSubjectsPerStage`, `conditionalPower`, `conditionalCriticalValue`, `overallEffects`, and `stDevH1`. The function has to contain the three-dots argument '...' (see examples).

Value

Returns a `SimulationResults` object. The following generics (R generic functions) are available for this object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

Examples

```
# Assess a treatment-arm selection strategy with three active arms,
# if the better of the arms is selected for the second stage, and
# compare it with the no-selection case.
# Assume a linear dose-response relationship
maxNumberOfIterations <- 100
designIN <- getDesignInverseNormal(typeOfDesign = "OF", kMax = 2)
```

```
sim <- getSimulationMultiArmMeans(design = designIN,
    activeArms = 3, typeOfShape = "linear",
    muMaxVector = seq(0,0.8,0.2),
    intersectionTest = "Simes",
    typeOfSelection = "best",
    plannedSubjects = c(30,60),
    maxNumberOfIterations = maxNumberOfIterations)

sim0 <- getSimulationMultiArmMeans(design = designIN,
    activeArms = 3, typeOfShape = "linear",
    muMaxVector = seq(0,0.8,0.2),
    intersectionTest = "Simes",
    typeOfSelection = "all",
    plannedSubjects = c(30,60),
    maxNumberOfIterations = maxNumberOfIterations)

sim$rejectAtLeastOne
sim$expectedNumberOfSubjects

sim0$rejectAtLeastOne
sim0$expectedNumberOfSubjects

# Compare the power of the conditional Dunnett test with the power of the
# combination test using Dunnett's intersection tests if no treatment arm
# selection takes place. Asseume a linear dose-response relationship.
maxNumberOfIterations <- 100
designIN <- getDesignInverseNormal(typeOfDesign = "asUser",
    userAlphaSpending = c(0, 0.025))
designCD <- getDesignConditionalDunnett(secondStageConditioning = TRUE)

index <- 1
for (design in c(designIN, designCD)) {
    results <- getSimulationMultiArmMeans(design, activeArms = 3,
        muMaxVector = seq(0, 1, 0.2), typeOfShape = "linear",
        plannedSubjects = cumsum(rep(20, 2)),
        intersectionTest = "Dunnett",
        typeOfSelection = "all", maxNumberOfIterations = maxNumberOfIterations)
    if (index == 1) {
        drift <- results$effectMatrix[nrow(results$effectMatrix), ]
        plot(drift, results$rejectAtLeastOne, type = "l", lty = 1,
            lwd = 3, col = "black", ylab = "Power")
    } else {
        lines(drift,results$rejectAtLeastOne, type = "l",
            lty = index, lwd = 3, col = "red")
    }
    index <- index + 1
}
legend("topleft", legend=c("Combination Dunnett", "Conditional Dunnett"),
    col=c("black", "red"), lty = (1:2), cex = 0.8)

# Assess the design characteristics of a user defined selection
# strategy in a two-stage design using the inverse normal method
# with constant bounds. Stopping for futility due to
```

```
# de-selection of all treatment arms.
designIN <- getDesignInverseNormal(typeOfDesign = "P", kMax = 2)

mySelection <- function(effectVector) {
    selectedArms <- (effectVector >= c(0, 0.1, 0.3))
    return(selectedArms)
}

results <- getSimulationMultiArmMeans(designIN, activeArms = 3,
    muMaxVector = seq(0, 1, 0.2),
    typeOfShape = "linear",
    plannedSubjects = c(30,60),
    intersectionTest = "Dunnett",
    typeOfSelection = "userDefined",
    selectArmsFunction = mySelection,
    maxNumberOfIterations = 100)

options(rpact.summary.output.size = "medium")
summary(results)
if (require(ggplot2)) plot(results, type = c(5,3,9), grid = 4)
```

---

getSimulationMultiArmRates
Get Simulation Multi-Arm Rates

Description

Returns the simulated power, stopping and selection probabilities, conditional power, and expected sample size for testing rates in a multi-arm treatment groups testing situation.

Usage

```
getSimulationMultiArmRates(
  design = NULL,
  ...,
  activeArms = 3L,
  effectMatrix = NULL,
  typeOfShape = c("linear", "sigmoidEmax", "userDefined"),
  piMaxVector = seq(0.2, 0.5, 0.1),
  piControl = 0.2,
  gED50 = NA_real_,
  slope = 1,
  intersectionTest = c("Dunnett", "Bonferroni", "Simes", "Sidak", "Hierarchical"),
  directionUpper = TRUE,
  adaptations = NA,
  typeOfSelection = c("best", "rBest", "epsilon", "all", "userDefined"),
  effectMeasure = c("effectEstimate", "testStatistic"),
```

```
  successCriterion = c("all", "atLeastOne"),
  epsilonValue = NA_real_,
  rValue = NA_real_,
  threshold = -Inf,
  plannedSubjects = NA_real_,
  allocationRatioPlanned = NA_real_,
  minNumberOfSubjectsPerStage = NA_real_,
  maxNumberOfSubjectsPerStage = NA_real_,
  conditionalPower = NA_real_,
  piH1 = NA_real_,
  piControlH1 = NA_real_,
  maxNumberOfIterations = 1000L,
  seed = NA_real_,
  calcSubjectsFunction = NULL,
  selectArmsFunction = NULL,
  showStatistics = FALSE
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`activeArms`	The number of active treatment arms to be compared with control, default is `3`.
`effectMatrix`	Matrix of effect sizes with `activeArms` columns and number of rows reflecting the different situations to consider.
`typeOfShape`	The shape of the dose-response relationship over the treatment groups. This can be either `"linear"`, `"sigmoidEmax"`, or `"userDefined"`. If `"sigmoidEmax"` is selected, `"gED50"` and `"slope"` has to be entered to specify the ED50 and the slope of the sigmoid Emax model. For `"linear"` and `"sigmoidEmax"`, `"muMaxVector"` specifies the range of effect sizes for the treatment group with highest response. If `"userDefined"` is selected, `"effectMatrix"` has to be entered.
`piMaxVector`	Range of assumed probabilities for the treatment group with highest response for `"linear"` and `"sigmoidEmax"` model, default is `seq(0, 1, 0.2)`.
`piControl`	If specified, the assumed probability in the control arm for simulation and under which the sample size recalculation is performed.
`gED50`	If `typeOfShape = "sigmoidEmax"` is selected, `"gED50"` has to be entered to specify the ED50 of the sigmoid Emax model.
`slope`	If `typeOfShape = "sigmoidEmax"` is selected, `"slope"` can be entered to specify the slope of the sigmoid Emax model, default is 1.
`intersectionTest`	Defines the multiple test for the intersection hypotheses in the closed system of hypotheses. Five options are available in multi-arm designs: `"Dunnett"`, `"Bonferroni"`, `"Simes"`, `"Sidak"`, and `"Hierarchical"`, default is `"Dunnett"`.

`directionUpper`	Specifies the direction of the alternative, only applicable for one-sided testing; default is `TRUE` which means that larger values of the test statistics yield smaller p-values.
`adaptations`	A vector of length `kMax - 1` indicating whether or not an adaptation takes place at interim k, default is `rep(TRUE, kMax - 1)`.
`typeOfSelection`	The way the treatment arms or populations are selected at interim. Five options are available: `"best"`, `"rbest"`, `"epsilon"`, `"all"`, and `"userDefined"`, default is `"best"`. For `"rbest"` (select the `rValue` best treatment arms/populations), the parameter `rValue` has to be specified, for `"epsilon"` (select treatment arm/population not worse than epsilon compared to the best), the parameter `epsilonValue` has to be specified. If `"userDefined"` is selected, `"selectArmsFunction"` or `"selectPopulationsFunction"` has to be specified.
`effectMeasure`	Criterion for treatment arm/population selection, either based on test statistic (`"testStatistic"`) or effect estimate (difference for means and rates or ratio for survival) (`"effectEstimate"`), default is `"effectEstimate"`.
`successCriterion`	Defines when the study is stopped for efficacy at interim. Two options are available: `"all"` stops the trial if the efficacy criterion is fulfilled for all selected treatment arms/populations, `"atLeastOne"` stops if at least one of the selected treatment arms/populations is shown to be superior to control at interim, default is `"all"`.
`epsilonValue`	For `typeOfSelection = "epsilon"` (select treatment arm / population not worse than epsilon compared to the best), the parameter `epsilonValue` has to be specified.
`rValue`	For `typeOfSelection = "rbest"` (select the `rValue` best treatment arms / populations), the parameter `rValue` has to be specified.
`threshold`	Selection criterion: treatment arm / population is selected only if `effectMeasure` exceeds `threshold`, default is `-Inf`. `threshold` can also be a vector of length `activeArms` referring to a separate threshold condition over the treatment arms.
`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. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs, `plannedSubjects` refers to the number of subjects per selected active arm.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`minNumberOfSubjectsPerStage`	When performing a data driven sample size recalculation, the vector `minNumberOfSubjectsPerStage` with length kMax determines the minimum number of subjects per stage (i.e., not cumulated), the first element is not taken into account. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs

	`minNumberOfSubjectsPerStage` refers to the minimum number of subjects per selected active arm.
`maxNumberOfSubjectsPerStage`	When performing a data driven sample size recalculation, the vector `maxNumberOfSubjectsPerStage` with length kMax determines the maximum number of subjects per stage (i.e., not cumulated), the first element is not taken into account. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs `maxNumberOfSubjectsPerStage` refers to the maximum number of subjects per selected active arm.
`conditionalPower`	If `conditionalPower` together with `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerSt` (or `minNumberOfEventsPerStage` and `maxNumberOfEventsPerStage` for survival designs) is specified, a sample size recalculation based on the specified conditional power is performed. It is defined as the power for the subsequent stage given the current data. By default, the conditional power will be calculated under the observed effect size. Optionally, you can also specify `thetaH1` and `stDevH1` (for simulating means), `pi1H1` and `pi2H1` (for simulating rates), or `thetaH1` (for simulating hazard ratios) as parameters under which it is calculated and the sample size recalculation is performed.
`piH1`	If specified, the assumed probability in the active treatment arm(s) under which the sample size recalculation is performed.
`piControlH1`	If specified, the assumed probability in the reference group (if different from `piControl`) for which the conditional power was calculated.
`maxNumberOfIterations`	The number of simulation iterations, default is `1000`.
`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 recalculation is performed with conditional power with specified `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerStage` (see details and examples).
`selectArmsFunction`	Optionally, a function can be entered that defines the way of how treatment arms are selected. This function is allowed to depend on `effectVector` with length `activeArms` and `stage` (see examples).
`showStatistics`	If `TRUE`, summary statistics of the simulated data are displayed for the `print` command, otherwise the output is suppressed, default is `FALSE`.

Details

At given design the function simulates the power, stopping probabilities, selection probabilities, and expected sample size at given number of subjects, parameter configuration, and treatment arm selection rule in the multi-arm situation. An allocation ratio can be specified referring to the ratio of number of subjects in the active treatment groups as compared to the control group.

The definition of `pi1H1` and/or `piControl` makes only sense if `kMax` > 1 and if `conditionalPower`, `minNumberOfSubjectsPerStage`, and `maxNumberOfSubjectsPerStage` (or `calcSubjectsFunction`) are defined.

`calcSubjectsFunction`
This function returns the number of subjects at given conditional power and conditional critical value for specified testing situation. The function might depend on the variables `stage`, `selectedArms`, `directionUpper`, `plannedSubjects`, `allocationRatioPlanned`, `minNumberOfSubjectsPerStage`, `maxNumberOfSubjectsPerStage`, `conditionalPower`, `conditionalCriticalValue`, `overallRates`, `overallRatesControl`, `piH1`, and `piControlH1`. The function has to contain the three-dots argument '...' (see examples).

Value

Returns a `SimulationResults` object. The following generics (R generic functions) are available for this object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

Examples

```
# Simulate the power of the combination test with two interim stages and
# O'Brien & Fleming boundaries using Dunnett's intersection tests if the
# best treatment arm is selected at first interim. Selection only take
# place if a non-negative treatment effect is observed (threshold = 0);
# 20 subjects per stage and treatment arm, simulation is performed for
# four parameter configurations.
maxNumberOfIterations <- 50
designIN <- getDesignInverseNormal(typeOfDesign = "OF")

effectMatrix <- matrix(c(0.2,0.2,0.2,
    0.4,0.4,0.4,
    0.4,0.5,0.5,
    0.4,0.5,0.6),
    byrow = TRUE, nrow = 4, ncol = 3)

x <- getSimulationMultiArmRates(design = designIN, typeOfShape = "userDefined",
    effectMatrix = effectMatrix , piControl = 0.2,
    typeOfSelection = "best", threshold = 0, intersectionTest = "Dunnett",
    plannedSubjects = c(20, 40, 60),
```

```
    maxNumberOfIterations = maxNumberOfIterations)

summary(x)
```

---

getSimulationMultiArmSurvival

Get Simulation Multi-Arm Survival

Description

Returns the simulated power, stopping and selection probabilities, conditional power, and expected sample size for testing hazard ratios in a multi-arm treatment groups testing situation. In contrast to `getSimulationSurvival()` (where survival times are simulated), normally distributed logrank test statistics are simulated.

Usage

```
getSimulationMultiArmSurvival(
  design = NULL,
  ...,
  activeArms = 3L,
  effectMatrix = NULL,
  typeOfShape = c("linear", "sigmoidEmax", "userDefined"),
  omegaMaxVector = seq(1, 2.6, 0.4),
  gED50 = NA_real_,
  slope = 1,
  intersectionTest = c("Dunnett", "Bonferroni", "Simes", "Sidak", "Hierarchical"),
  directionUpper = TRUE,
  adaptations = NA,
  typeOfSelection = c("best", "rBest", "epsilon", "all", "userDefined"),
  effectMeasure = c("effectEstimate", "testStatistic"),
  successCriterion = c("all", "atLeastOne"),
  correlationComputation = c("alternative", "null"),
  epsilonValue = NA_real_,
  rValue = NA_real_,
  threshold = -Inf,
  plannedEvents = NA_real_,
  allocationRatioPlanned = NA_real_,
  minNumberOfEventsPerStage = NA_real_,
  maxNumberOfEventsPerStage = NA_real_,
  conditionalPower = NA_real_,
  thetaH1 = NA_real_,
  maxNumberOfIterations = 1000L,
  seed = NA_real_,
  calcEventsFunction = NULL,
```

```
  selectArmsFunction = NULL,
  showStatistics = FALSE
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`activeArms`	The number of active treatment arms to be compared with control, default is `3`.
`effectMatrix`	Matrix of effect sizes with `activeArms` columns and number of rows reflecting the different situations to consider.
`typeOfShape`	The shape of the dose-response relationship over the treatment groups. This can be either `"linear"`, `"sigmoidEmax"`, or `"userDefined"`. If `"sigmoidEmax"` is selected, `"gED50"` and `"slope"` has to be entered to specify the ED50 and the slope of the sigmoid Emax model. For `"linear"` and `"sigmoidEmax"`, `"muMaxVector"` specifies the range of effect sizes for the treatment group with highest response. If `"userDefined"` is selected, `"effectMatrix"` has to be entered.
`omegaMaxVector`	Range of hazard ratios with highest response for `"linear"` and `"sigmoidEmax"` model, default is `seq(1, 2.6, 0.4)`.
`gED50`	If `typeOfShape = "sigmoidEmax"` is selected, `"gED50"` has to be entered to specify the ED50 of the sigmoid Emax model.
`slope`	If `typeOfShape = "sigmoidEmax"` is selected, `"slope"` can be entered to specify the slope of the sigmoid Emax model, default is 1.
`intersectionTest`	Defines the multiple test for the intersection hypotheses in the closed system of hypotheses. Five options are available in multi-arm designs: `"Dunnett"`, `"Bonferroni"`, `"Simes"`, `"Sidak"`, and `"Hierarchical"`, default is `"Dunnett"`.
`directionUpper`	Specifies the direction of the alternative, only applicable for one-sided testing; default is `TRUE` which means that larger values of the test statistics yield smaller p-values.
`adaptations`	A vector of length `kMax - 1` indicating whether or not an adaptation takes place at interim k, default is `rep(TRUE, kMax - 1)`.
`typeOfSelection`	The way the treatment arms or populations are selected at interim. Five options are available: `"best"`, `"rbest"`, `"epsilon"`, `"all"`, and `"userDefined"`, default is `"best"`. For `"rbest"` (select the `rValue` best treatment arms/populations), the parameter `rValue` has to be specified, for `"epsilon"` (select treatment arm/population not worse than epsilon compared to the best), the parameter `epsilonValue` has to be specified. If `"userDefined"` is selected, `"selectArmsFunction"` or `"selectPopulationsFunction"` has to be specified.

`effectMeasure`	Criterion for treatment arm/population selection, either based on test statistic (`"testStatistic"`) or effect estimate (difference for means and rates or ratio for survival) (`"effectEstimate"`), default is `"effectEstimate"`.
`successCriterion`	Defines when the study is stopped for efficacy at interim. Two options are available: `"all"` stops the trial if the efficacy criterion is fulfilled for all selected treatment arms/populations, `"atLeastOne"` stops if at least one of the selected treatment arms/populations is shown to be superior to control at interim, default is `"all"`.
`correlationComputation`	If `correlationComputation = "alternative"`, for simulating log-rank statistics in the many-to-one design, a correlation matrix according to Deng et al. (Biometrics, 2019) accounting for the respective alternative is used; if `correlationComputation = "null"`, a constant correlation matrix valid under the null, i.e., not accounting for the alternative is used, default is `"alternative"`.
`epsilonValue`	For `typeOfSelection = "epsilon"` (select treatment arm / population not worse than epsilon compared to the best), the parameter `epsilonValue` has to be specified.
`rValue`	For `typeOfSelection = "rbest"` (select the `rValue` best treatment arms / populations), the parameter `rValue` has to be specified.
`threshold`	Selection criterion: treatment arm / population is selected only if `effectMeasure` exceeds `threshold`, default is `-Inf`. `threshold` can also be a vector of length `activeArms` referring to a separate threshold condition over the treatment arms.
`plannedEvents`	`plannedEvents` is a vector of length `kMax` (the number of stages of the design) that determines the number of cumulated (overall) events in survival designs when the interim stages are planned. For two treatment arms, it is the number of events for both treatment arms. For multi-arm designs, `plannedEvents` refers to the overall number of events for the selected arms plus control.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`minNumberOfEventsPerStage`	When performing a data driven sample size recalculation, the vector `minNumberOfEventsPerStage` with length kMax determines the minimum number of events per stage (i.e., not cumulated), the first element is not taken into account.
`maxNumberOfEventsPerStage`	When performing a data driven sample size recalculation, the vector `maxNumberOfEventsPerStage` with length kMax determines the maximum number of events per stage (i.e., not cumulated), the first element is not taken into account.
`conditionalPower`	If `conditionalPower` together with `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerSt` (or `minNumberOfEventsPerStage` and `maxNumberOfEventsPerStage` for survival designs) is specified, a sample size recalculation based on the specified conditional power is performed. It is defined as the power for the subsequent stage given the current data. By default, the conditional power will be calculated under the observed effect size. Optionally, you can also specify `thetaH1`

	and `stDevH1` (for simulating means), `pi1H1` and `pi2H1` (for simulating rates), or `thetaH1` (for simulating hazard ratios) as parameters under which it is calculated and the sample size recalculation is performed.
`thetaH1`	If specified, the value of the alternative under which the conditional power or sample size recalculation calculation is performed.
`maxNumberOfIterations`	The number of simulation iterations, default is `1000`.
`seed`	The seed to reproduce the simulation, default is a random seed.
`calcEventsFunction`	Optionally, a function can be entered that defines the way of performing the sample size recalculation. By default, sample size recalculation is performed with conditional power with specified `minNumberOfEventsPerStage` and `maxNumberOfEventsPerStage` (see details and examples).
`selectArmsFunction`	Optionally, a function can be entered that defines the way of how treatment arms are selected. This function is allowed to depend on `effectVector` with length `activeArms` and `stage` (see examples).
`showStatistics`	If `TRUE`, summary statistics of the simulated data are displayed for the `print` command, otherwise the output is suppressed, default is `FALSE`.

Details

At given design the function simulates the power, stopping probabilities, selection probabilities, and expected sample size at given number of subjects, parameter configuration, and treatment arm selection rule in the multi-arm situation. An allocation ratio can be specified referring to the ratio of number of subjects in the active treatment groups as compared to the control group.

The definition of `thetaH1` makes only sense if `kMax` > 1 and if `conditionalPower`, `minNumberOfEventsPerStage`, and `maxNumberOfEventsPerStage` (or `calcEventsFunction`) are defined.

`calcEventsFunction`
This function returns the number of events at given conditional power and conditional critical value for specified testing situation. The function might depend on the variables `stage`, `selectedArms`, `plannedEvents`, `directionUpper`, `allocationRatioPlanned`, `minNumberOfEventsPerStage`, `maxNumberOfEventsPerSt` `conditionalPower`, `conditionalCriticalValue`, and `overallEffects`. The function has to contain the three-dots argument '...' (see examples).

Value

Returns a `SimulationResults` object. The following generics (R generic functions) are available for this object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

Examples

```
# Assess different selection rules for a two-stage survival design with
# O'Brien & Fleming alpha spending boundaries and (non-binding) stopping
# for futility if the test statistic is negative.
# Number of events at the second stage is adjusted based on conditional
# power 80% and specified minimum and maximum number of Events.
maxNumberOfIterations <- 50
design <- getDesignInverseNormal(typeOfDesign = "asOF", futilityBounds = 0)

y1 <- getSimulationMultiArmSurvival(design = design, activeArms = 4,
    intersectionTest = "Simes", typeOfShape = "sigmoidEmax",
    omegaMaxVector = seq(1, 2, 0.5), gED50 = 2, slope = 4,
    typeOfSelection = "best", conditionalPower = 0.8,
    minNumberOfEventsPerStage = c(NA_real_, 30),
    maxNumberOfEventsPerStage = c(NA_real_, 90),
    maxNumberOfIterations = maxNumberOfIterations,
    plannedEvents = c(75, 120))

y2 <- getSimulationMultiArmSurvival(design = design, activeArms = 4,
    intersectionTest = "Simes", typeOfShape = "sigmoidEmax",
    omegaMaxVector = seq(1,2,0.5), gED50 = 2, slope = 4,
    typeOfSelection = "epsilon", epsilonValue = 0.2,
    effectMeasure = "effectEstimate",
    conditionalPower = 0.8, minNumberOfEventsPerStage = c(NA_real_, 30),
    maxNumberOfEventsPerStage = c(NA_real_, 90),
    maxNumberOfIterations = maxNumberOfIterations,
   plannedEvents = c(75, 120))

y1$effectMatrix

y1$rejectAtLeastOne
y2$rejectAtLeastOne

y1$selectedArms
y2$selectedArms
```

`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,
  normalApproximation = TRUE,
  riskRatio = FALSE,
  thetaH0 = ifelse(riskRatio, 1, 0),
  pi1 = seq(0.2, 0.5, 0.1),
  pi2 = NA_real_,
  plannedSubjects = NA_real_,
  directionUpper = TRUE,
  allocationRatioPlanned = NA_real_,
  minNumberOfSubjectsPerStage = NA_real_,
  maxNumberOfSubjectsPerStage = NA_real_,
  conditionalPower = NA_real_,
  pi1H1 = NA_real_,
  pi2H1 = NA_real_,
  maxNumberOfIterations = 1000L,
  seed = NA_real_,
  calcSubjectsFunction = NULL,
  showStatistics = FALSE
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to 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`.
`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 exact test of Fisher (two samples) is used for calculating the p-values. In the survival setting `normalApproximation = FALSE` has no effect.
`riskRatio`	If `TRUE`, the design characteristics for one-sided testing of H0: `pi1 / pi2 = thetaH0` are simulated, default is `FALSE`.
`thetaH0`	The null hypothesis value, default is `0` for the normal and the binary case (testing means and rates, respectively), it is `1` for the survival case (testing the hazard

ratio).

For non-inferiority designs, `thetaH0` is the non-inferiority bound. That is, in case of (one-sided) testing of

- *means*: a value `!= 0` (or a value `!= 1` for testing the mean ratio) can be specified.
- *rates*: a value `!= 0` (or a value `!= 1` for testing the risk ratio `pi1 / pi2`) can be specified.
- *survival data*: a bound for testing H0: `hazard ratio = thetaH0 != 1` can be specified.

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`.

`pi1`	A numeric value or vector that represents 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)` (power calculations and simulations) or `seq(0.4, 0.6, 0.1)` (sample size calculations).
`pi2`	A numeric value that represents 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. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs, `plannedSubjects` refers to the number of subjects per selected active arm.
`directionUpper`	Specifies the direction of the alternative, only applicable for one-sided testing; default is `TRUE` which means that larger values of the test statistics yield smaller p-values.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`minNumberOfSubjectsPerStage`	When performing a data driven sample size recalculation, the vector `minNumberOfSubjectsPerStage` with length kMax determines the minimum number of subjects per stage (i.e., not cumulated), the first element is not taken into account. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs `minNumberOfSubjectsPerStage` refers to the minimum number of subjects per selected active arm.
`maxNumberOfSubjectsPerStage`	When performing a data driven sample size recalculation, the vector `maxNumberOfSubjectsPerStage` with length kMax determines the maximum number of subjects per stage (i.e., not cumulated), the first element is not taken into account. For two treatment arms, it is the number of subjects for both treatment arms. For multi-arm designs `maxNumberOfSubjectsPerStage` refers to the maximum number of subjects per selected active arm.

`conditionalPower`	If `conditionalPower` together with `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerSt` (or `minNumberOfEventsPerStage` and `maxNumberOfEventsPerStage` for survival designs) is specified, a sample size recalculation based on the specified conditional power is performed. It is defined as the power for the subsequent stage given the current data. By default, the conditional power will be calculated under the observed effect size. Optionally, you can also specify `thetaH1` and `stDevH1` (for simulating means), `pi1H1` and `pi2H1` (for simulating rates), or `thetaH1` (for simulating hazard ratios) as parameters under which it is calculated and 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.
`maxNumberOfIterations`	The number of simulation iterations, default is `1000`.
`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 recalculation is performed with conditional power with specified `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerStage` (see details and examples).
`showStatistics`	If `TRUE`, summary statistics of the simulated data are displayed for the `print` command, otherwise the output is suppressed, default is `FALSE`.

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.

The definition of `pi1H1` and/or `pi2H1` makes only sense if `kMax` > 1 and if `conditionalPower`, `minNumberOfSubjectsPerStage`, and `maxNumberOfSubjectsPerStage` (or `calcSubjectsFunction`) are defined.

`calcSubjectsFunction`
This function returns the number of subjects at given conditional power and conditional critical value for specified testing situation. The function might depend on variables `stage`, `riskRatio`, `thetaH0`, `groups`, `plannedSubjects`, `sampleSizesPerStage`, `directionUpper`, `allocationRatioPlanned`, `minNumberOfSubjectsPerStage`, `maxNumberOfSubjectsPerStage`, `conditionalPower`, `conditionalCriticalValue`, `overallRate`, `farringtonManningValue1`, and `farringtonManningValue2`. The function has to contain the three-dots argument '...' (see examples).

Value

Returns a `SimulationResults` object. The following generics (R generic functions) are available for this object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

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. `overallRate1`: The cumulative rate in treatment group 1.
11. `overallRate2`: The cumulative rate in treatment group 2.
12. `stagewiseRates1`: The stage-wise rate in treatment group 1.

13. `stagewiseRates2`: The stage-wise rate in treatment group 2.
14. `sampleSizesPerStage1`: The stage-wise sample size in treatment group 1.
15. `sampleSizesPerStage2`: The stage-wise 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`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

Examples

```
# Fixed sample size design (two groups) with total sample
# size 120, pi1 = (0.3,0.4,0.5,0.6) and pi2 = 0.3
getSimulationRates(pi1 = seq(0.3, 0.6, 0.1), pi2 = 0.3,
    plannedSubjects = 120, maxNumberOfIterations = 10)

# 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 (cumulative) rates and specified minNumberOfSubjectsPerStage
# and maxNumberOfSubjectsPerStage

# 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,
        minNumberOfSubjectsPerStage,
        maxNumberOfSubjectsPerStage,
        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(
            minNumberOfSubjectsPerStage[stage],
            stageSubjects), maxNumberOfSubjectsPerStage[stage]))
        return(stageSubjects)
    }
}
getSimulationRates(designIN, pi1 = seq(0.3, 0.6, 0.1), pi2 = 0.3,
    plannedSubjects = c(40, 80), minNumberOfSubjectsPerStage = c(40, 20),
    maxNumberOfSubjectsPerStage = 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 = 1,
  directionUpper = TRUE,
  pi1 = NA_real_,
  pi2 = NA_real_,
  lambda1 = NA_real_,
  lambda2 = NA_real_,
  median1 = NA_real_,
  median2 = NA_real_,
  hazardRatio = NA_real_,
  kappa = 1,
  piecewiseSurvivalTime = NA_real_,
  allocation1 = 1,
  allocation2 = 1,
```

```
  eventTime = 12,
  accrualTime = c(0, 12),
  accrualIntensity = 0.1,
  accrualIntensityType = c("auto", "absolute", "relative"),
  dropoutRate1 = 0,
  dropoutRate2 = 0,
  dropoutTime = 12,
  maxNumberOfSubjects = NA_real_,
  plannedEvents = NA_real_,
  minNumberOfEventsPerStage = NA_real_,
  maxNumberOfEventsPerStage = NA_real_,
  conditionalPower = NA_real_,
  thetaH1 = NA_real_,
  maxNumberOfIterations = 1000L,
  maxNumberOfRawDatasetsPerStage = 0,
  longTimeSimulationAllowed = FALSE,
  seed = NA_real_,
  showStatistics = FALSE
)
```

Arguments

`design`	The trial design. If no trial design is specified, a fixed sample size design is used. In this case, Type I error rate `alpha`, Type II error rate `beta`, `twoSidedPower`, and `sided` can be directly entered as argument where necessary.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`thetaH0`	The null hypothesis value, default is `0` for the normal and the binary case (testing means and rates, respectively), it is `1` for the survival case (testing the hazard ratio). For non-inferiority designs, `thetaH0` is the non-inferiority bound. That is, in case of (one-sided) testing of • *means*: a value `!= 0` (or a value `!= 1` for testing the mean ratio) can be specified. • *rates*: a value `!= 0` (or a value `!= 1` for testing the risk ratio `pi1 / pi2`) can be specified. • *survival data*: a bound for testing H0: `hazard ratio = thetaH0 != 1` can be specified. 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`.
`directionUpper`	Specifies the direction of the alternative, only applicable for one-sided testing; default is `TRUE` which means that larger values of the test statistics yield smaller p-values.
`pi1`	A numeric value or vector that represents the assumed event rate in the treatment group, default is `seq(0.2, 0.5, 0.1)` (power calculations and simulations) or `seq(0.4, 0.6, 0.1)` (sample size calculations).

`pi2`	A numeric value that represents 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).
`median1`	The assumed median survival time in the treatment group, there is no default.
`median2`	The assumed median survival time in the reference group, there is no default.
`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, there is no default.
`kappa`	A numeric value > 0. A `kappa != 1` will be used for the specification of the shape of the Weibull distribution. Default is `1`, i.e., the exponential survival distribution is used instead of the Weibull distribution. Note that the Weibull distribution cannot be used for the piecewise definition of the survival time distribution, i.e., only `piecewiselambda` (as a single value) and `kappa` can be specified. This function is equivalent to `pweibull(t, shape = kappa, scale = 1 / lambda)` of the `stats` package, 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.
`piecewiseSurvivalTime`	A vector that specifies the time intervals for the piecewise definition of the exponential survival time cumulative distribution function (for details see `getPiecewiseSurvivalTime`).
`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 intervals for the study, default is `c(0, 12)` (for details see `getAccrualTime`).
`accrualIntensity`	A vector of accrual intensities, default is the relative intensity `0.1` (for details see `getAccrualTime`).
`accrualIntensityType`	A character value specifying the accrual intensity input type. Must be one of `"auto"`, `"absolute"`, or `"relative"`; default is `"auto"`, i.e., if all values are $<$ 1 the type is `"relative"`, otherwise it is `"absolute"`.
`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) that determines the number of cumulated (overall) events in survival designs when the interim stages are planned. For two treatment arms, it is the number of events for both treatment arms. For multi-arm designs, `plannedEvents` refers to the overall number of events for the selected arms plus control.
`minNumberOfEventsPerStage`	When performing a data driven sample size recalculation, the vector `minNumberOfEventsPerStage` with length kMax determines the minimum number of events per stage (i.e., not cumulated), the first element is not taken into account.
`maxNumberOfEventsPerStage`	When performing a data driven sample size recalculation, the vector `maxNumberOfEventsPerStage` with length kMax determines the maximum number of events per stage (i.e., not cumulated), the first element is not taken into account.
`conditionalPower`	If `conditionalPower` together with `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerSt` (or `minNumberOfEventsPerStage` and `maxNumberOfEventsPerStage` for survival designs) is specified, a sample size recalculation based on the specified conditional power is performed. It is defined as the power for the subsequent stage given the current data. By default, the conditional power will be calculated under the observed effect size. Optionally, you can also specify `thetaH1` and `stDevH1` (for simulating means), `pi1H1` and `pi2H1` (for simulating rates), or `thetaH1` (for simulating hazard ratios) as parameters under which it is calculated and the sample size recalculation is performed.
`thetaH1`	If specified, the value of the alternative under which the conditional power or sample size recalculation calculation is performed.
`maxNumberOfIterations`	The number of simulation iterations, default is `1000`.
`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.
`showStatistics`	If `TRUE`, summary statistics of the simulated data are displayed for the `print` command, otherwise the output is suppressed, default is `FALSE`.

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. More precisely, unequal randomization ratios must be specified via the two integer arguments `allocation1` and `allocation2` which describe how many subjects are consecutively enrolled in each group, respectively, before a subject is assigned to the other group. For example, the arguments `allocation1 = 2`, `allocation2 = 1`, `maxNumberOfSubjects = 300` specify 2:1 randomization with 200 subjects randomized to intervention and 100 to control. (Caveat: Do not use `allocation1 = 200`, `allocation2 = 100`, `maxNumberOfSubjects = 300` as this would imply that the 200 intervention subjects are enrolled prior to enrollment of any control subjects.)

`conditionalPower`
The definition of `thetaH1` makes only sense if `kMax` > 1 and if `conditionalPower`, `minNumberOfEventsPerStage`, and `maxNumberOfEventsPerStage` are defined.

Note that `numberOfSubjects`, `numberOfSubjects1`, and `numberOfSubjects2` in the output are expected number of subjects.

Value

Returns a `SimulationResults` object. The following generics (R generic functions) are available for this object:

- `names` to obtain the field names,
- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

Piecewise survival time

The first element of the vector `piecewiseSurvivalTime` 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).

Staggered patient entry

`accrualTime` is the time period of subjects' accrual in a study. It can be a value that defines the end of accrual or a vector. In this case, `accrualTime` can be used to define a non-constant accrual over time. For this, `accrualTime` is a vector that defines the accrual intervals. The first element of `accrualTime` must be equal to `0` and, additionally, `accrualIntensity` needs to be specified. `accrualIntensity` itself is a value or a vector (depending on the length of `accrualtime`) that defines the intensity how subjects enter the trial in the intervals defined through `accrualTime`.

`accrualTime` can also be a list that combines the definition of the accrual time and accrual intensity (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), `maxNumberOfSubjects > 0` needs to be specified and the end of accrual is calculated. In that case, `accrualIntensity` is the number of subjects per time unit, i.e., the absolute accrual intensity.

If the length of `accrualTime` equals the length of `accrualIntensity - 1` (i.e., the end of accrual is defined), `maxNumberOfSubjects` is calculated if the absolute accrual intensity is given. If all elements in `accrualIntensity` are smaller than 1, `accrualIntensity` defines the *relative* intensity how subjects enter the trial. For example, `accrualIntensity = c(0.1, 0.2)` specifies that in the second accrual interval the intensity is doubled as compared to the first accrual interval. The actual (absolute) accrual intensity is calculated for the calculated or given `maxNumberOfSubjects`. Note that the default is `accrualIntensity = 0.1` meaning that the *absolute* accrual intensity will be calculated.

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, plannedEvents = 30)`
`simulationResults$show(showStatistics = FALSE)`

Example 2:
`simulationResults <- getSimulationSurvival(maxNumberOfSubjects = 100, plannedEvents = 30)`
`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.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

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 = 10)

# 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 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
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 Fleming 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),
    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
getSimulationSurvival(median1 = 5, median2 = 3, plannedEvents = 40,
    maxNumberOfSubjects = 200, directionUpper = FALSE,
    maxNumberOfIterations = 50)

# Specify effect size based on median survival
# times of Weibull distribtion with kappa = 2
getSimulationSurvival(median1 = 5, median2 = 3, kappa = 2,
    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 minNumberOfEventsPerStage and
# maxNumberOfEventsPerStage 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),
    minNumberOfEventsPerStage = c(NA, 44, 44),
    maxNumberOfEventsPerStage = 4 * c(NA, 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),
    minNumberOfEventsPerStage = c(NA, 44, 44),
    maxNumberOfEventsPerStage = 4 * c(NA, 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),
    minNumberOfEventsPerStage = c(NA, 44, 44),
    maxNumberOfEventsPerStage = 4 * c(NA, 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, ..., stage = NA_integer_)
```

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` and should be created with the function `getDataset`. For more information see `getDataset`.
`...`	Further (optional) arguments to be passed: `thetaH0` The null hypothesis value, default is `0` for the normal and the binary case (testing means and rates, respectively), it is 1 for the survival case (testing the hazard ratio). For non-inferiority designs, `thetaH0` is the non-inferiority bound. That is, in case of (one-sided) testing of

- *means*: a value `!= 0` (or a value `!= 1` for testing the mean ratio) can be specified.
- *rates*: a value `!= 0` (or a value `!= 1` for testing the risk ratio `pi1 / pi2`) can be specified.
- *survival data*: a bound for testing H0: `hazard ratio = thetaH0 != 1` can be specified.

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`.

`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 exact 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 `TRUE`.

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

`intersectionTest` Defines the multiple test for the intersection hypotheses in the closed system of hypotheses when testing multiple hypotheses. Five options are available in multi-arm designs: `"Dunnett"`, `"Bonferroni"`, `"Simes"`, `"Sidak"`, and `"Hierarchical"`, default is `"Dunnett"`. Four options are available in population enrichment designs: `"SpiessensDebois"` (one subset only), `"Bonferroni"`, `"Simes"`, and `"Sidak"`, default is `"Simes"`.

`varianceOption` Defines the way to calculate the variance in multiple treatment arms (> 2) or population enrichment designs for testing means. For multiple arms, three options are available: `"overallPooled"`, `"pairwisePooled"`, and `"notPooled"`, default is `"overallPooled"`. For enrichment designs, the options are: `"pooled"`, `"pooledFromFull"` (one subset only), and `"notPooled"`, default is `"pooled"`.

`stratifiedAnalysis` For enrichment designs, typically a stratified analysis should be chosen. For testing means and rates, also a non-stratified analysis based on overall data can be performed. For survival data, only a stratified analysis is possible (see Brannath et al., 2009), default is `TRUE`.

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

Details

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

Value

Returns a `StageResults` object.

- `names` to obtain the field names,

- `print` to print the object,
- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

How to get help for generic functions

Click on the link of a generic in the list above to go directly to the help documentation of the `rpact` specific implementation of the generic. Note that you can use the R function `methods` to get all the methods of a generic and to identify the object specific name of it, e.g., use `methods("plot")` to get all the methods for the `plot` generic. There you can find, e.g., `plot.AnalysisResults` and obtain the specific help documentation linked above by typing `?plot.AnalysisResults`.

See Also

Other analysis functions: `getAnalysisResults()`, `getClosedCombinationTestResults()`, `getClosedConditionalDunn` `getConditionalPower()`, `getConditionalRejectionProbabilities()`, `getFinalConfidenceInterval()`, `getFinalPValue()`, `getRepeatedConfidenceIntervals()`, `getRepeatedPValues()`, `getTestActions()`

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(stageResults, ...)
```

Arguments

`stageResults`	The results at given stage, obtained from `getStageResults`.
`...`	Only available for backward compatibility.

Details

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

Value

Returns a `character` vector of length `kMax` Returns a `numeric` vector of length `kMax`containing the test actions of each stage.

See Also

Other analysis functions: `getAnalysisResults()`, `getClosedCombinationTestResults()`, `getClosedConditionalDunn` `getConditionalPower()`, `getConditionalRejectionProbabilities()`, `getFinalConfidenceInterval()`, `getFinalPValue()`, `getRepeatedConfidenceIntervals()`, `getRepeatedPValues()`, `getStageResults()`

Examples

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

`kable` *Create tables in Markdown*

Description

The `kable()` function returns a single table for a single object that inherits from class `ParameterSet`.

Usage

```
kable(x, ...)
```

Arguments

`x`	The object that inherits from `ParameterSet`.
`...`	Other arguments (see `kable`).

Details

Generic to represent a parameter set in Markdown.

`kable.ParameterSet` *Create output in Markdown*

Description

The `kable()` function returns the output of the specified object formatted in Markdown.

Usage

```
kable.ParameterSet(x, ...)
```

Arguments

`x`	A `ParameterSet`. If x does not inherit from class `ParameterSet`, `knitr::kable(x)` will be returned.
`...`	Other arguments (see `kable`).

Details

Generic function to represent a parameter set in Markdown. Use `options("rpact.print.heading.base.number" = "NUMBER")` (where `NUMBER` is an integer value >= -1) to specify the heading level. The default is `options("rpact.print.heading.base.number" = "0")`, i.e., the top headings start with ## in Markdown. `options("rpact.print.heading.base.number" = "-1")` means that all headings will be written bold but are not explicit defined as header.

`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_,
  allocationRatioPlanned = NA_real_,
  main = NA_character_,
  xlab = NA_character_,
  ylab = NA_character_,
```

```
  legendTitle = NA_character_,
  palette = "Set1",
  legendPosition = NA_integer_,
  showSource = FALSE,
  grid = 1,
  plotSettings = NULL
)
```

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 plot 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, `assumedStDev` (assumed standard deviation) can be specified (default is `1`). • `piTreatmentRange`: 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`: Specifies the direction of the alternative, only applicable for one-sided testing; default is `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`.
`type`	The plot type (default = 1). Note that at the moment only one type (the conditional power plot) is available.
`nPlanned`	The additional (i.e., "new" and not cumulative) sample size planned for each of the subsequent stages. The argument must be a vector with length equal to the number of remaining stages and contain the combined sample size from both treatment groups if two groups are considered. For survival outcomes, it should contain the planned number of additional events. For multi-arm designs, it is the per-comparison (combined) sample size. For enrichment designs, it is the (combined) sample size for the considered sub-population.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`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: • `-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 the base R `plot` function. Alternatively `showSource` can be defined as one of the following character values: • `"commands"`: returns a character vector with plot commands • `"axes"`: returns a list with the axes definitions • `"test"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function does not stop if an error occurs) • `"validate"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function stops if an error occurs) Note: no plot object will be returned if `showSource` is a character.
`grid`	An integer value specifying the output of multiple plots. By default (`1`) a list of `ggplot` objects will be returned. If a `grid` value > 1 was specified, a grid plot will be returned if the number of plots is <= specified `grid` value; a list of `ggplot` objects will be returned otherwise. If `grid = 0` is specified, all plots will be created using `print` command and a list of `ggplot` objects will be returned invisible. Note that one of the following packages must be installed to create a grid plot: 'ggpubr', 'gridExtra', or 'cowplot'.
`plotSettings`	An object of class `PlotSettings` created by `getPlotSettings`.

Details

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

Value

Returns a `ggplot2` object.

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,
  plotSettings = NULL
)
```

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 plot arguments. At the moment `xlim` and `ylim` are implemented for changing x or y axis limits without dropping data observations.
`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 the base R `plot` function. Alternatively `showSource` can be defined as one of the following character values: • `"commands"`: returns a character vector with plot commands • `"axes"`: returns a list with the axes definitions • `"test"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function does not stop if an error occurs) • `"validate"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function stops if an error occurs) Note: no plot object will be returned if `showSource` is a character.
`plotSettings`	An object of class `PlotSettings` created by `getPlotSettings`.

Details

Generic function to plot all kinds of datasets.

Value

Returns a `ggplot2` object.

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.EventProbabilities`

Event Probabilities Plotting

Description

Plots an object that inherits from class `EventProbabilities`.

Usage

```
## S3 method for class 'EventProbabilities'
plot(
  x,
  y,
  ...,
  allocationRatioPlanned = x$allocationRatioPlanned,
  main = NA_character_,
  xlab = NA_character_,
  ylab = NA_character_,
  type = 1L,
  legendTitle = NA_character_,
  palette = "Set1",
  plotPointsEnabled = NA,
  legendPosition = NA_integer_,
  showSource = FALSE,
  plotSettings = NULL
)
```

Arguments

`x`	The object that inherits from `EventProbabilities`.
`y`	An optional object that inherits from `NumberOfSubjects`.
`...`	Optional plot arguments. At the moment `xlim` and `ylim` are implemented for changing x or y axis limits without dropping data observations.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`main`	The main title.
`xlab`	The x-axis label.
`ylab`	The y-axis label.
`type`	The plot type (default = 1). Note that at the moment only one type is available.
`legendTitle`	The legend title, default is `""`.
`palette`	The palette, default is `"Set1"`.

`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 the base R `plot` function. Alternatively `showSource` can be defined as one of the following character values: • `"commands"`: returns a character vector with plot commands • `"axes"`: returns a list with the axes definitions • `"test"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function does not stop if an error occurs) • `"validate"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function stops if an error occurs) Note: no plot object will be returned if `showSource` is a character.
`plotSettings`	An object of class `PlotSettings` created by `getPlotSettings`.

Details

Generic function to plot an event probabilities object.

Generic function to plot a parameter set.

Value

Returns a `ggplot2` object.

`plot.NumberOfSubjects` *Number Of Subjects Plotting*

Description

Plots an object that inherits from class `NumberOfSubjects`.

Usage

```
## S3 method for class 'NumberOfSubjects'
plot(
  x,
  y,
  ...,
  allocationRatioPlanned = NA_real_,
  main = NA_character_,
  xlab = NA_character_,
  ylab = NA_character_,
  type = 1L,
  legendTitle = NA_character_,
  palette = "Set1",
  plotPointsEnabled = NA,
  legendPosition = NA_integer_,
  showSource = FALSE,
  plotSettings = NULL
)
```

Arguments

`x`	The object that inherits from `NumberOfSubjects`.
`y`	An optional object that inherits from `EventProbabilities`.
`...`	Optional plot arguments. At the moment `xlim` and `ylim` are implemented for changing x or y axis limits without dropping data observations.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. Will be ignored if `y` is undefined.
`main`	The main title.
`xlab`	The x-axis label.
`ylab`	The y-axis label.
`type`	The plot type (default = 1). Note that at the moment only one type is available.
`legendTitle`	The legend title, default is `""`.
`palette`	The palette, default is `"Set1"`.
`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 the base R `plot` function. Alternatively `showSource` can be defined as one of the following character values: - `"commands"`: returns a character vector with plot commands - `"axes"`: returns a list with the axes definitions - `"test"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function does not stop if an error occurs) - `"validate"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function stops if an error occurs) Note: no plot object will be returned if `showSource` is a character.
`plotSettings`	An object of class `PlotSettings` created by `getPlotSettings`.

Details

Generic function to plot an "number of subjects" object.

Generic function to plot a parameter set.

Value

Returns a `ggplot2` object.

`plot.ParameterSet` *Parameter Set Plotting*

Description

Plots an object that inherits from class `ParameterSet`.

Usage

```
## S3 method for class 'ParameterSet'
plot(
  x,
  y,
  ...,
  main = NA_character_,
  xlab = NA_character_,
  ylab = NA_character_,
  type = 1L,
  palette = "Set1",
```

```
  legendPosition = NA_integer_,
  showSource = FALSE,
  plotSettings = NULL
)
```

Arguments

`x`	The object that inherits from `ParameterSet`.
`y`	Not available for this kind of plot (is only defined to be compatible to the generic plot function).
`...`	Optional plot arguments. At the moment `xlim` and `ylim` are implemented for changing x or y axis limits without dropping data observations.
`main`	The main title.
`xlab`	The x-axis label.
`ylab`	The y-axis label.
`type`	The plot type (default = 1).
`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: • `-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 the base R `plot` function. Alternatively `showSource` can be defined as one of the following character values: • `"commands"`: returns a character vector with plot commands • `"axes"`: returns a list with the axes definitions • `"test"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function does not stop if an error occurs) • `"validate"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function stops if an error occurs) Note: no plot object will be returned if `showSource` is a character.
`plotSettings`	An object of class `PlotSettings` created by `getPlotSettings`.

Details

Generic function to plot a parameter set.

Value

Returns a `ggplot2` object.

`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 = 1L,
  palette = "Set1",
  theta = seq(-1, 1, 0.01),
  plotPointsEnabled = NA,
  legendPosition = NA_integer_,
  showSource = FALSE,
  grid = 1,
  plotSettings = NULL
)
```

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).
`...`	Optional plot arguments. At the moment `xlim` and `ylim` are implemented for changing x or y axis limits without dropping data observations.
`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 'Overall Success' plot (multi-arm and enrichment only) • `2`: creates a 'Success per Stage' plot (multi-arm and enrichment only) • `3`: creates a 'Selected Arms per Stage' plot (multi-arm and enrichment only) • `4`: creates a 'Reject per Stage' or 'Rejected Arms per Stage' plot • `5`: creates a 'Overall Power and Early Stopping' plot • `6`: creates a 'Expected Number of Subjects and Power / Early Stop' or 'Expected Number of Events and Power / Early Stop' plot • `7`: creates an 'Overall Power' plot • `8`: creates an 'Overall Early Stopping' plot • `9`: creates an 'Expected Sample Size' or 'Expected Number of Events' plot • `10`: creates a 'Study Duration' plot (non-multi-arm and non-enrichment survival only) • `11`: creates an 'Expected Number of Subjects' plot (non-multi-arm and non-enrichment survival only) • `12`: creates an 'Analysis Times' plot (non-multi-arm and non-enrichment survival only) • `13`: creates a 'Cumulative Distribution Function' plot (non-multi-arm and non-enrichment survival only) • `14`: creates a 'Survival Function' plot (non-multi-arm and non-enrichment survival only) • `"all"`: creates all available plots and returns it as a grid plot or list
`palette`	The palette, default is `"Set1"`.
`theta`	A vector of standardized effect sizes (theta values), default is a sequence from -1 to 1.
`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 the base R `plot` function. Alternatively `showSource` can be defined as one of the following character values:

- `"commands"`: returns a character vector with plot commands
- `"axes"`: returns a list with the axes definitions
- `"test"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function does not stop if an error occurs)
- `"validate"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function stops if an error occurs)

Note: no plot object will be returned if `showSource` is a character.

`grid`	An integer value specifying the output of multiple plots. By default (`1`) a list of `ggplot` objects will be returned. If a `grid` value > 1 was specified, a grid plot will be returned if the number of plots is <= specified `grid` value; a list of `ggplot` objects will be returned otherwise. If `grid = 0` is specified, all plots will be created using `print` command and a list of `ggplot` objects will be returned invisible. Note that one of the following packages must be installed to create a grid plot: 'ggpubr', 'gridExtra', or 'cowplot'.
`plotSettings`	An object of class `PlotSettings` created by `getPlotSettings`.

Details

Generic function to plot all kinds of simulation results.

Value

Returns a `ggplot2` object.

Examples

```
results <- getSimulationMeans(alternative = 0:4, stDev = 5,
    plannedSubjects = 40, maxNumberOfIterations = 1000)
plot(results, type = 5)
```

`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,
  allocationRatioPlanned = 1,
  main = NA_character_,
  xlab = NA_character_,
  ylab = NA_character_,
  legendTitle = NA_character_,
  palette = "Set1",
  legendPosition = NA_integer_,
  showSource = FALSE,
  plotSettings = NULL
)
```

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 plot 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). • `piTreatmentRange`: 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`: Specifies the direction of the alternative, only applicable for one-sided testing; default is `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.
`type`	The plot type (default = 1). Note that at the moment only one type (the conditional power plot) is available.
`nPlanned`	The additional (i.e., "new" and not cumulative) sample size planned for each of the subsequent stages. The argument must be a vector with length equal to the number of remaining stages and contain the combined sample size from both treatment groups if two groups are considered. For survival outcomes, it should contain the planned number of additional events. For multi-arm designs, it is the per-comparison (combined) sample size. For enrichment designs, it is the (combined) sample size for the considered sub-population.
`allocationRatioPlanned`	The planned allocation ratio `n1 / n2` for a two treatment groups design, default is `1`. For multi-arm designs, it is the allocation ratio relating the active arm(s) to the control.
`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: • `-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 the base R `plot` function. Alternatively `showSource` can be defined as one of the following character values: • `"commands"`: returns a character vector with plot commands • `"axes"`: returns a list with the axes definitions • `"test"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function does not stop if an error occurs) • `"validate"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function stops if an error occurs) Note: no plot object will be returned if `showSource` is a character.
`plotSettings`	An object of class `PlotSettings` created by `getPlotSettings`.

Details

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

Value

Returns a `ggplot2` object.

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.SummaryFactory` *Summary Factory Plotting*

Description

Plots a summary factory.

Usage

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

Arguments

`x`	The summary factory object.
`y`	Not available for this kind of plot (is only defined to be compatible to the generic plot function).
`...`	Optional plot arguments. At the moment `xlim` and `ylim` are implemented for changing x or y axis limits without dropping data observations.

Details

Generic function to plot all kinds of summary factories.

Value

Returns a `ggplot2` object.

`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 = 1L,
  palette = "Set1",
  theta = seq(-1, 1, 0.01),
  nMax = NA_integer_,
  plotPointsEnabled = NA,
  legendPosition = NA_integer_,
  showSource = FALSE,
  grid = 1,
  plotSettings = NULL
)
```

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).
`...`	Optional plot arguments. At the moment `xlim` and `ylim` are implemented for changing x or y axis limits without dropping data observations.
`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 '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 • `"all"`: creates all available plots and returns it as a grid plot or list
`palette`	The palette, default is `"Set1"`.
`theta`	A vector of standardized effect sizes (theta values), default is a sequence from -1 to 1.
`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 the base R `plot` function. Alternatively `showSource` can be defined as one of the following character values: • `"commands"`: returns a character vector with plot commands • `"axes"`: returns a list with the axes definitions • `"test"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function does not stop if an error occurs) • `"validate"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function stops if an error occurs) Note: no plot object will be returned if `showSource` is a character.
`grid`	An integer value specifying the output of multiple plots. By default (`1`) a list of `ggplot` objects will be returned. If a `grid` value > 1 was specified, a grid plot will be returned if the number of plots is <= specified `grid` value; a list of `ggplot` objects will be returned otherwise. If `grid = 0` is specified, all plots will be created using `print` command and a list of `ggplot` objects will be returned invisible. Note that one of the following packages must be installed to create a grid plot: 'ggpubr', 'gridExtra', or 'cowplot'.
`plotSettings`	An object of class `PlotSettings` created by `getPlotSettings`.

Details

Generic function to plot a trial design.

Generic function to plot a trial design.

Note that `nMax` is not an argument that it passed to `ggplot2`. Rather, the underlying calculations (e.g. power for different theta's or average sample size) are based on calls to function `getPowerAndAverageSampleNumber` which has argument `nMax`. I.e., `nMax` is not an argument to ggplot2 but to `getPowerAndAverageSampleNumber` which is called prior to plotting.

Value

Returns a `ggplot2` object.

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, 5L, 1L),
  palette = "Set1",
  theta = seq(-1, 1, 0.01),
  plotPointsEnabled = NA,
  legendPosition = NA_integer_,
  showSource = FALSE,
  grid = 1,
  plotSettings = NULL
)
```

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).
`...`	Optional plot arguments. At the moment `xlim` and `ylim` are implemented for changing x or y axis limits without dropping data observations.
`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 '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 • `"all"`: creates all available plots and returns it as a grid plot or list
`palette`	The palette, default is `"Set1"`.

`theta`	A vector of standardized effect sizes (theta values), default is a sequence from -1 to 1.
`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 the base R `plot` function. Alternatively `showSource` can be defined as one of the following character values: • `"commands"`: returns a character vector with plot commands • `"axes"`: returns a list with the axes definitions • `"test"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function does not stop if an error occurs) • `"validate"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function stops if an error occurs) Note: no plot object will be returned if `showSource` is a character.
`grid`	An integer value specifying the output of multiple plots. By default (`1`) a list of `ggplot` objects will be returned. If a `grid` value > 1 was specified, a grid plot will be returned if the number of plots is <= specified `grid` value; a list of `ggplot` objects will be returned otherwise. If `grid = 0` is specified, all plots will be created using `print` command and a list of `ggplot` objects will be returned invisible. Note that one of the following packages must be installed to create a grid plot: 'ggpubr', 'gridExtra', or 'cowplot'.
`plotSettings`	An object of class `PlotSettings` created by `getPlotSettings`.

Details

Generic function to plot all kinds of trial design plans.

Value

Returns a `ggplot2` object.

Examples

```
if (require(ggplot2)) plot(getSampleSizeMeans())
```

`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,
  grid = 1,
  plotSettings = NULL
)
```

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).
`...`	Optional plot arguments. At the moment `xlim` and `ylim` are implemented for changing x or y axis limits without dropping data observations.
`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 '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 • `"all"`: creates all available plots and returns it as a grid plot or list
`main`	The main title.
`xlab`	The x-axis label.
`ylab`	The y-axis label.
`palette`	The palette, default is `"Set1"`.
`theta`	A vector of standardized effect sizes (theta values), default is a sequence from -1 to 1.
`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 the base R `plot` function. Alternatively `showSource` can be defined as one of the following character values: • `"commands"`: returns a character vector with plot commands • `"axes"`: returns a list with the axes definitions • `"test"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function does not stop if an error occurs) • `"validate"`: all plot commands will be validated with `eval(parse())` and returned as character vector (function stops if an error occurs) Note: no plot object will be returned if `showSource` is a character.
`grid`	An integer value specifying the output of multiple plots. By default (`1`) a list of `ggplot` objects will be returned. If a `grid` value > 1 was specified, a grid plot will be returned if the number of plots is <= specified `grid` value; a list of `ggplot` objects will be returned otherwise. If `grid = 0` is specified, all plots will

	be created using `print` command and a list of `ggplot` objects will be returned invisible. Note that one of the following packages must be installed to create a grid plot: 'ggpubr', 'gridExtra', or 'cowplot'.
`plotSettings`	An object of class `PlotSettings` created by `getPlotSettings`.

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)
```

`plotTypes` *Get Available Plot Types*

Description

Function to identify the available plot types of an object.

Usage

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

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 `getDesignGroupSequential` 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

`plotTypes` and `getAvailablePlotTypes` are equivalent, i.e., `plotTypes` is a short form of `getAvailablePlotTypes`.

`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

Value

Depending on how the `output` is specified, a numeric vector, a character vector, or a list will be returned.

Examples

```
design <- getDesignInverseNormal(kMax = 2)
getAvailablePlotTypes(design, "numeric")
plotTypes(design, "caption")
getAvailablePlotTypes(design, "numcap")
plotTypes(design, "capnum")
```

`rawDataTwoArmNormal` *Raw Dataset Of A Two Arm Continuous Outcome With Covariates*

Description

An artificial dataset that was randomly generated with simulated normal data. The data set has six variables:

1. Subject id 2. Stage number 3. Group name 4. An example outcome in that we are interested in 5. The first covariate *gender* 6. The second covariate *covariate*

Usage

```
rawDataTwoArmNormal
```

Format

A `data.frame` object.

Details

See the vignette "Two-arm analysis for continuous data with covariates from raw data" to learn how to

* import raw data from a csv file, * calculate estimated adjusted (marginal) means (EMMs, least-squares means) for a linear model, and * perform two-arm interim analyses with these data.

You can use `rawDataTwoArmNormal` to reproduce the examples in the vignette.

`rcmd` *Get Object R Code*

Description

Returns the R source command of a result object.

Usage

```
rcmd(
  obj,
  ...,
  leadingArguments = NULL,
  includeDefaultParameters = FALSE,
  stringWrapParagraphWidth = 90,
  prefix = "",
  postfix = "",
  stringWrapPrefix = "",
  newArgumentValues = list()
)

getObjectRCode(
  obj,
  ...,
  leadingArguments = NULL,
  includeDefaultParameters = FALSE,
  stringWrapParagraphWidth = 90,
  prefix = "",
  postfix = "",
  stringWrapPrefix = "",
  newArgumentValues = list(),
  tolerance = 1e-07
)
```

Arguments

`obj`	The result object.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`leadingArguments`	A character vector with arguments that shall be inserted at the beginning of the function command, e.g., `design = x`. Be careful with this option because the created R command may no longer be valid if used.
`includeDefaultParameters`	If `TRUE`, default parameters will be included in all `rpact` commands; default is `FALSE`.
`stringWrapParagraphWidth`	An integer value defining the number of characters after which a line break shall be inserted; set to `NULL` to insert no line breaks.
`prefix`	A character string that shall be added to the beginning of the R command.
`postfix`	A character string that shall be added to the end of the R command.
`stringWrapPrefix`	A prefix character string that shall be added to each new line, typically some spaces.
`newArgumentValues`	A named list with arguments that shall be renewed in the R command, e.g., `newArgumentValues = list(informationRates = c(0.5, 1))`.
`tolerance`	The tolerance for defining a value as default.

Details

`getObjectRCode` (short: `rcmd`) recreates the R commands that result in the specified object `obj`. `obj` must be an instance of class `ParameterSet`.

Value

A `character` value or vector will be returned.

`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. The following generics (R generic functions) are available for this result object:

- `names` to obtain the field names,
- `print` to print the object,

- `summary` to display a summary of the object,
- `plot` to plot the object,
- `as.data.frame` to coerce the object to a `data.frame`,
- `as.matrix` to coerce the object to a `matrix`.

See Also

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

Examples

```
dataFileRates <- system.file("extdata",
    "dataset_rates.csv",
    package = "rpact"
)
if (dataFileRates != "") {
    datasetRates <- readDataset(dataFileRates)
    datasetRates
}

dataFileMeansMultiArm <- system.file("extdata",
    "dataset_means_multi-arm.csv",
    package = "rpact"
)
if (dataFileMeansMultiArm != "") {
    datasetMeansMultiArm <- readDataset(dataFileMeansMultiArm)
    datasetMeansMultiArm
}

dataFileRatesMultiArm <- system.file("extdata",
    "dataset_rates_multi-arm.csv",
    package = "rpact"
)
if (dataFileRatesMultiArm != "") {
    datasetRatesMultiArm <- readDataset(dataFileRatesMultiArm)
    datasetRatesMultiArm
}

dataFileSurvivalMultiArm <- system.file("extdata",
    "dataset_survival_multi-arm.csv",
    package = "rpact"
)
if (dataFileSurvivalMultiArm != "") {
    datasetSurvivalMultiArm <- readDataset(dataFileSurvivalMultiArm)
    datasetSurvivalMultiArm
}
```

`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.

Examples

```
dataFile <- system.file("extdata", "datasets_rates.csv", package = "rpact")
if (dataFile != "") {
    datasets <- readDatasets(dataFile)
    datasets
}
```

`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, simulation, 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:10.1007/9783319325620)

See Also

Useful links:

- `https://www.rpact.org`
- `https://www.rpact.com`
- `https://github.com/rpact-com/rpact`
- Report bugs at `https://github.com/rpact-com/rpact/issues`

`setOutputFormat` *Set Output Format*

Description

With this function the format of the standard outputs of all `rpact` objects can be changed and set user defined respectively.

Usage

```
setOutputFormat(
  parameterName = NA_character_,
  ...,
  digits = NA_integer_,
  nsmall = NA_integer_,
  trimSingleZeros = NA,
  futilityProbabilityEnabled = NA,
  file = NA_character_,
  resetToDefault = FALSE,
  roundFunction = NA_character_
)
```

Arguments

`parameterName`	The name of the parameter whose output format shall be edited. Leave the default `NA_character_` if the output format of all parameters shall be edited.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.

`digits`	How many significant digits are to be used for a numeric value. The default, `NULL`, uses getOption("digits"). Allowed values are `0` <= `digits` <= `20`.
`nsmall`	The minimum number of digits to the right of the decimal point in formatting real numbers in non-scientific formats. Allowed values are `0` <= `nsmall` <= `20`.
`trimSingleZeros`	If `TRUE` zero values will be trimmed in the output, e.g., "0.00" will displayed as "0"
`futilityProbabilityEnabled`	If `TRUE` very small value (< 1e-09) will be displayed as "0", default is `FALSE`.
`file`	An optional file name of an existing text file that contains output format definitions (see Details for more information).
`resetToDefault`	If `TRUE` all output formats will be reset to default value. Note that other settings will be executed afterwards if specified, default is `FALSE`.
`roundFunction`	A character value that specifies the R base round function to use, default is `NA_character_`. Allowed values are "ceiling", "floor", "trunc", "round", "signif", and `NA_character_`.

Details

Output formats can be written to a text file (see `getOutputFormat`). To load your personal output formats read a formerly saved file at the beginning of your work with `rpact`, e.g. execute `setOutputFormat(file = "my_rpact_output_formats.txt")`.

Note that the `parameterName` must not match exactly, e.g., for p-values the following parameter names will be recognized amongst others:

1. `p value`
2. `p.values`
3. `p-value`
4. `pValue`
5. `rpact.output.format.p.value`

See Also

`format` for details on the function used internally to format the values.

Other output formats: `getOutputFormat()`

Examples

```
# show output format of p values
getOutputFormat("p.value")

# set new p value output format
setOutputFormat("p.value", digits = 5, nsmall = 5)

# show sample sizes as smallest integers not less than the not rounded values
setOutputFormat("sample size", digits = 0, nsmall = 0, roundFunction = "ceiling")
getSampleSizeMeans()
```

```
# show sample sizes as smallest integers not greater than the not rounded values
setOutputFormat("sample size", digits = 0, nsmall = 0, roundFunction = "floor")
getSampleSizeMeans()

# set new sample size output format without round function
setOutputFormat("sample size", digits = 2, nsmall = 2)
getSampleSizeMeans()

# reset sample size output format to default
setOutputFormat("sample size")
getSampleSizeMeans()
getOutputFormat("sample size")
```

`testPackage` *Test Package*

Description

This function allows the installed package `rpact` to be tested.

Usage

```
testPackage(
  outDir = ".",
  ...,
  completeUnitTestSetEnabled = TRUE,
  types = "tests",
  connection = list(token = NULL, secret = NULL)
)
```

Arguments

`outDir`	The output directory where all test results shall be saved. By default the current working directory is used.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`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", "vignettes")`, default is "tests" only.
`connection`	A `list` where owners of the rpact validation documentation can enter a `token` and a `secret` to get full access to all unit tests, e.g., to fulfill regulatory requirements (see www.rpact.com for more information).

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`.

Value

The value of `completeUnitTestSetEnabled` will be returned invisible.

Examples

```
## Not run:
testPackage()

## End(Not run)
```

`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 (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`kappa`	A numeric value > 0. A `kappa != 1` will be used for the specification of the shape of the Weibull distribution. Default is `1`, i.e., the exponential survival distribution is used instead of the Weibull distribution. Note that the Weibull distribution cannot be used for the piecewise definition of the survival time distribution, i.e., only `piecewiselambda` (as a single value) and `kappa` can be specified. This function is equivalent to `pweibull(t, shape = kappa, scale = 1 / lambda)` of the `stats` package, 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, e.g., 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 calculate probabilities, quantiles, or random numbers. In this case, no piecewise definition is possible, i.e., only piecewiseLambda (as a single value) and kappa need to be specified.

Value

A `numeric` value or vector will be returned.

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 = 12, kappa = 1)

getLambdaByMedian(median, kappa = 1)

getHazardRatioByPi(pi1, pi2, eventTime = 12, kappa = 1)

getPiByLambda(lambda, eventTime = 12, kappa = 1)

getPiByMedian(median, eventTime = 12, kappa = 1)

getMedianByLambda(lambda, kappa = 1)

getMedianByPi(piValue, eventTime = 12, 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`	A numeric value > 0. A `kappa != 1` will be used for the specification of the shape of the Weibull distribution. Default is `1`, i.e., the exponential survival distribution is used instead of the Weibull distribution. Note that the Weibull distribution cannot be used for the piecewise definition of the survival time distribution, i.e., only `piecewiselambda` (as a single value) and `kappa` can be specified. This function is equivalent to `pweibull(t, shape = kappa, scale = 1 / lambda)` of the `stats` package, 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.

Details

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

Value

Returns a `numeric` value or vector will be returned.

`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.

Examples

```
## Not run:
datasetOfRates <- getDataset(
    n1 = c(11, 13, 12, 13),
    n2 = c(8, 10, 9, 11),
    events1 = c(10, 10, 12, 12),
```

```
    events2 = c(3, 5, 5, 6)
)
writeDataset(datasetOfRates, "dataset_rates.csv")

## End(Not run)
```

---

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.

Examples

```
## Not run:
d1 <- getDataset(
    n1 = c(11, 13, 12, 13),
    n2 = c(8, 10, 9, 11),
    events1 = c(10, 10, 12, 12),
    events2 = c(3, 5, 5, 6)
)
d2 <- getDataset(
    n1 = c(9, 13, 12, 13),
    n2 = c(6, 10, 9, 11),
    events1 = c(10, 10, 12, 12),
    events2 = c(4, 5, 5, 6)
)
datasets <- list(d1, d2)
writeDatasets(datasets, "datasets_rates.csv")

## End(Not run)
```

Index