Package 'rpact'

November 17, 2026

Title Confirmatory Adaptive Clinical Trial Design and Analysis

Version 5.0.0

Date 2026-11-16

Description Design, simulation, and analysis of confirmatory clinical trials with continuous, binary, survival, and count data endpoints according to the methods described in the monograph by Wassmer and Brannath (2025) <doi:10.1007/978-3-031-89669-9>. This includes sample size and power calculations for standard fixed-sample and classical group sequential designs, as well as power simulations for innovative trial designs. The latter include multi-stage adaptive hypothesis tests based on the combination testing principle, adaptive procedures such as sample size reassessment, and methods for multi-arm multi-stage (MAMS) and population enrichment trials.

License LGPL-3

Encoding UTF-8

LazyData true

URL https://www.rpact.org,
https://www.rpact.com,
https://docs.rpact.org,
https://github.com/rpact-com/rpact,
https://demo.rpact.cloud

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

Language en-US

Depends R (>= 4.1.0)

Imports grid,
grDevices,
parallel,
methods,
stats,
utils,
graphics,
tools,
rlang,
R6 (>= 2.5.1),
knitr (>= 1.19),
Rcpp (>= 1.0.3)

LinkingTo Rcpp

Suggests callr,
parallelly,
ggplot2 (>= 3.5.0),
testthat (>= 3.0.0),
rmarkdown (>= 1.10),
rappdirs (>= 0.3.3)

VignetteBuilder knitr, rmarkdown

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first f_analysis_multiarm_means_inverse_normal,
f_analysis_multiarm_means_fisher,
f_analysis_multiarm_means_inverse_normal_direction_lower,
f_analysis_multiarm_means_fisher_direction_lower,
f_design_optimal_user,
class_analysis_dataset,
f_analysis_enrichment_means,
f_analysis_base_rates,
f_design_plan_survival,
f_analysis_multiarm_rates,
f_design_optimal_background,
f_simulation_multiarm_rates,
f_analysis_base_means,
f_simulation_multiarm_means,
f_simulation_multiarm_survival_patient_wise_compare_basic,
f_analysis_multiarm_means_kmax1,
f_analysis_multiarm_means_fixed,
f_simulation_plot,
f_simulation_multiarm_survival

Collate 'RcppExports.R'
'f_logger.R'
'class_dictionary.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_core_plot.R'
'class_design.R'
'f_object_r_code.R'
'f_analysis_base.R'
'class_analysis_dataset.R'
'class_analysis_stage_results.R'
'class_analysis_results.R'
'class_design_optimal.R'
'f_design_general_utilities.R'
'class_time.R'

'class_design_set.R'
'class_design_plan.R'
'class_design_power_and_asn.R'
'class_event_probabilities.R'
'class_fisher_information.R'
'f_simulation_base_counts.R'
'f_simulation_utilities.R'
'f_simulation_base_survival.R'
'class_simulation_results.R'
'class_performance_score.R'
'class_summary.R'
'class_time_accrual.R'
'class_time_piecewise_survival.R'
'f_error_handling.R'
'f_piecewise_survival_time_utilities.R'
'class_time_piecewise_survival_multigroup.R'
'class_trial_comparison.R'
'data.R'
'f_analysis_base_general.R'
'f_analysis_base_means.R'
'f_analysis_base_rates.R'
'f_analysis_base_survival.R'
'f_analysis_boundary_recalculation.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_as251.R'
'f_comparison.R'
'f_comparison_adapters.R'
'f_comparison_parameters.R'
'f_comparison_plot.R'
'f_comparison_r_code.R'
'f_comparison_table.R'
'f_core_output_formats.R'
'f_design_fisher_combination_test.R'
'f_design_fixed.R'
'f_design_futility_bounds.R'
'f_design_group_sequential.R'
'parameter_descriptions.R'
'f_design_optimal_background.R'
'f_design_optimal_user.R'
'f_design_optimal_utilities.R'
'f_design_plan_boundary_data.R'
'f_design_plan_counts.R'
'f_design_plan_means.R'
'f_design_plan_plot.R'
'f_design_plan_rates.R'

'f_design_plan_survival.R'
'f_design_plan_utilities.R'
'f_io_utilities.R'
'f_piecewise_survival_time.R'
'f_piecewise_survival_time_multiarm.R'
'f_quality_assurance.R'
'f_simulation_base_means.R'
'f_simulation_base_rates.R'
'f_simulation_calc_subjects_function.R'
'f_simulation_enrichment.R'
'f_simulation_enrichment_means.R'
'f_simulation_enrichment_rates.R'
'f_simulation_enrichment_survival_crude.R'
'f_simulation_enrichment_survival_function.R'
'f_simulation_enrichment_survival_pw.R'
'f_simulation_enrichment_survival_pw_basic.R'
'f_simulation_multiarm.R'
'f_simulation_multiarm_means.R'
'f_simulation_multiarm_rates.R'
'f_simulation_multiarm_survival_crude.R'
'f_simulation_multiarm_survival_function.R'
'f_simulation_multiarm_survival_pw.R'
'f_simulation_multiarm_survival_pw_basic.R'
'f_simulation_performance_score.R'
'f_simulation_piecewise_survival.R'
'f_simulation_plot.R'
'f_summary.R'
'f_summary_header.R'
'f_summary_parameters.R'
'f_summary_simulation_dropout.R'
'f_summary_title.R'
'f_summary_utilities.R'
'f_test_execution.R'
'f_test_reporting.R'
'f_warning_handling.R'
'parameter_column_names.R'
'parameter_formats.R'
'parameter_names.R'
'parameter_order.R'
'pkgname.R'

Config/roxygen2/version 8.1.1

Contents

add 7
as.data.frame.PiecewiseSurvivalTimeMultiGroup 7
as.data.frame.TrialComparison 8
as251Normal 10
as251StudentT 11
compare 12
delete 13

fetch.TrialComparison . . . 14
getAccrualTime . . . 15
getAnalysisResults . . . 17
getClosedCombinationTestResults . . . 23
getClosedConditionalDunnettTestResults . . . 25
getComparison . . . 26
getComparisonParameterDescriptions . . . 30
getComparisonParameterNames . . . 31
getComparisonPlotParameters . . . 32
getConditionalPower . . . 33
getConditionalRejectionProbabilities . . . 36
getData . . . 37
getDataset . . . 38
getDesignCharacteristics . . . 44
getDesignConditionalDunnett . . . 45
getDesignFisher . . . 47
getDesignFixed . . . 49
getDesignGroupSequential . . . 51
getDesignInverseNormal . . . 54
getDesignSet . . . 58
getEventProbabilities . . . 60
getFinalConfidenceInterval . . . 62
getFinalPValue . . . 64
getFisherInformation . . . 65
getFutilityBounds . . . 68
getGroupSequentialProbabilities . . . 74
getNumberOfSubjects . . . 76
getObjects . . . 78
getObservedInformationRates . . . 79
getOutputFormat . . . 80
getPerformanceScore . . . 82
getPiecewiseHazardRatioArray . . . 83
getPiecewiseSurvivalTime . . . 84
getPiecewiseSurvivalTimeEnrichment . . . 86
getPiecewiseSurvivalTimeMultiArm . . . 88
getPowerAndAverageSampleNumber . . . 89
getPowerCounts . . . 91
getPowerMeans . . . 94
getPowerRates . . . 97
getPowerSurvival . . . 100
getRawData . . . 106
getRepeatedConfidenceIntervals . . . 107
getRepeatedPValues . . . 109
getSampleSizeCounts . . . 110
getSampleSizeMeans . . . 113
getSampleSizeRates . . . 116
getSampleSizeSurvival . . . 119
getSimulationCounts . . . 125
getSimulationEnrichmentMeans . . . 129
getSimulationEnrichmentRates . . . 135
getSimulationEnrichmentSurvival . . . 141
getSimulationMeans . . . 150

getSimulationMultiArmMeans . . . 155
getSimulationMultiArmRates . . . 163
getSimulationMultiArmSurvival . . . 169
getSimulationRates . . . 178
getSimulationSurvival . . . 184
getStageResults . . . 196
getTestActions . . . 198
knit_print.SummaryFactory . . . 199
knit_print.TrialComparisonTable . . . 200
mvnprd . . . 201
mvstud . . . 202
obtain . . . 203
plot.AnalysisResults . . . 204
plot.Dataset . . . 206
plot.EventProbabilities . . . 208
plot.NumberOfSubjects . . . 210
plot.ParameterSet . . . 212
plot.SimulationResults . . . 213
plot.StageResults . . . 216
plot.SummaryFactory . . . 218
plot.TrialComparison . . . 219
plot.TrialDesign . . . 220
plot.TrialDesignPlan . . . 223
plot.TrialDesignSet . . . 225
plot.TrialDesignSummaries . . . 228
plotTypes . . . 228
print.SummaryFactory . . . 230
print.TrialComparison . . . 230
print.TrialDesignCharacteristics . . . 231
print.TrialDesignSummaries . . . 232
rcmd . . . 232
readDataset . . . 234
readDatasets . . . 236
rpact . . . 237
setOutputFormat . . . 238
summary.TrialComparison . . . 240
testPackage . . . 243
TrialComparison . . . 245
utilitiesForPiecewiseExponentialDistribution . . . 246
writeDataset . . . 248
writeDatasets . . . 249
[.TrialComparisonTable . . . 251
[[.TrialComparison . . . 251
%vs% . . . 252

Index **254**

add *Add Objects to a Collection*

Description

S3 generic for adding objects. The `TrialComparison` method delegates to `$add()` and modifies the existing comparison by reference.

The `TrialComparison` method appends objects to the comparison.

Usage

```
add(x, ...)

## S3 method for class 'TrialComparison'
add(x, ...)
```

Arguments

`x`	An object supporting an `add()` method, such as a `TrialComparison`.
`...`	For `TrialComparison`, designs, plans, simulations, lists or comparisons to append. The optional named argument `objects` is appended after the other arguments, with the same rules as `getComparison()`.

Value

The modified comparison for `TrialComparison`.

See Also

`getComparison()`, `delete()`

Examples

```
comparison <- getComparison(fixed = getDesignFixed())
comparison |> add(sequential = getDesignGroupSequential(kMax = 2))
comparison |> add(objects = list(other = getDesignFixed(alpha = 0.05)))
```

as.data.frame.PiecewiseSurvivalTimeMultiGroup *Coerce a Multi-Group Piecewise Survival Specification to a Data Frame*

Description

Returns one row per group, interval, and situation, including control and active hazards.

Usage

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

Arguments

x	A multi-arm or enrichment piecewise survival specification.
row.names	Optional row names.
optional	Unused; provided for compatibility with as.data.frame().
...	Further arguments, checked for unknown arguments.
niceColumnNamesEnabled	Whether descriptive column names should be used.
includeAllParameters	Unused; all specification rows are always included.

Value

A data frame in group-by-interval-by-situation order.

---

as.data.frame.TrialComparison

Trial Comparison Data Frame

Description

Converts a trial comparison to a data frame, optionally prepared for reporting.

Usage

```
## S3 method for class 'TrialComparison'
as.data.frame(
  x,
  row.names = NULL,
  optional = FALSE,
  ...,
  niceColumnNamesEnabled = FALSE,
  includeAllParameters = FALSE,
  harmonizeColumnNames = TRUE,
  removeConstantParameters = FALSE,
  transposed = FALSE,
  removeVariableParameters = FALSE,
  digits = NA_integer_,
```

```
  ceilingEnabled = FALSE,
  output = "data.frame"
)
```

Arguments

x	A TrialComparison.
row.names	Optional row names for the result.
optional	Unused; included for compatibility with as.data.frame().
...	Further arguments. Currently unused.
niceColumnNamesEnabled	Logical. If TRUE, nice looking column names will be used; syntactic names (variable names) otherwise (see make.names).
includeAllParameters	If TRUE, include all parameters supported by the existing data-frame methods of the individual objects.
harmonizeColumnNames	If TRUE (default), apply the class-specific mappings described below. If FALSE, retain original parameter names and omit extra fields that are included only to support these mappings. Readable labels remain controlled by niceColumnNamesEnabled.
removeConstantParameters	If TRUE, retain identifiers and only parameters with different object profiles, using the same rules as summary().
transposed	If TRUE, return parameters as rows and comparison records as columns, labelled by object and available stage/situation coordinates. Shared scalar profiles use Value; duplicate labels receive unique suffixes. If exactly one supplied or default input vector matches all situations across stages, its caption replaces Situation (for example, Alternative 1). Ambiguous axes retain Situation; shared profiles require agreement across objects. Mixed column types are converted to a common type by transposition.
removeVariableParameters	If TRUE, return shared parameter profiles once, with stage and situation coordinates where needed, without object identifiers. Cannot be combined with removeConstantParameters = TRUE.
digits	Integer number of decimal places for numeric parameters, or NA to retain full precision. Parameter profiles are compared before rounding.
ceilingEnabled	If TRUE, round sample-size and event-count parameters upwards to integers, using rpact parameter metadata. Variances are excluded.
output	Output mode: "data.frame" (default), "auto", "html", or "pdf". Reporting modes add class TrialComparisonTable and rendering attributes; they still return a data frame. "auto" selects HTML or LaTeX during knitting. "pdf" produces LaTeX tables for a PDF document, not a PDF file.

Details

With harmonizeColumnNames = TRUE, fixed-plan sample size columns are harmonized when the comparison includes interim analyses: nFixed, nFixed1, and nFixed2 become numberOfSubjects, numberOfSubjects1, and numberOfSubjects2. Fixed survival events use cumulativeEventsPerStage in mixed comparisons with interim analyses. Available sample-size H1 expectations use expectedNumberOfSubjects

and expectedNumberOfEvents, referring to each scenario's supplied alternative. See getComparison() for the class-specific rules and reference values.

Shared profiles use the same exact comparison as summary.TrialComparison(), applied to the selected harmonized or original columns. The summary's harmonizeColumnNames argument can be set independently of this data-frame option. Filtering precedes rounding, readable labels and transposition. Common scalar parameters are returned once; common stage or situation profiles retain their coordinates. Reporting modes render HTML in a horizontally scrollable container or LaTeX longtables split into column blocks with repeated identifier columns. LaTeX output requires the suggested rmarkdown package and registers the longtable dependency through knitr metadata. Other knitting targets use ordinary data-frame output. Reporting does not modify the comparison or its contained objects.

Multi-arm survival simulations use categoryType and category coordinates. comparison rows (for example, Treatment 1 vs. Control) contain stage-wise and cumulative event counts for treatment plus control. arm rows contain individual-arm events, selections, rejections and drop-out summaries, including a separate Control category. Arm-independent simulation results occur on overall / All arms rows. Input and design settings remain available for grouping. Result cells outside their applicable category are NA. The unfiltered data frame retains this expanded structure for plotting and grouping. Parameter filters use semantically normalized profiles: design settings do not acquire an arm dimension from repeated rows, whereas arm-level results retain their group domain even if all values are equal. Shared output combines the observed coordinate domains of the common profiles, without constructing new combinations or taking the first object's row layout. Scalars and lower-dimensional profiles are repeated only on compatible observed coordinates. Shared scalar-only output omits coordinate columns.

Value

A data.frame, unrounded and with all columns by default. Reporting modes additionally inherit from TrialComparisonTable for automatic knitr output.

---

as251Normal *Algorithm AS 251: Normal Distribution*

Description

Calculates the Multivariate Normal Distribution with Product Correlation Structure published by Charles Dunnett, Algorithm AS 251.1 Appl.Statist. (1989), Vol.38, No.3, doi:10.2307/2347754.

Usage

```
as251Normal(
  lower,
  upper,
  sigma,
  ...,
  eps = 1e-06,
  errorControl = c("strict", "halvingIntervals"),
  intervalSimpsonsRule = 0
)
```

Arguments

lower	Lower limits of integration. Array of N dimensions
upper	Upper limits of integration. Array of N dimensions
sigma	Values defining correlation structure. Array of N dimensions
...	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
eps	desired accuracy. Defaults to 1e-06
errorControl	error control. If set to "strict" (default), strict error control based on fourth derivative is used. If set to "halvingIntervals", error control based on halving intervals is used
intervalSimpsonsRule	Interval width for Simpson's rule. Value of zero caused a default .24 to be used

Details

For a multivariate normal vector with correlation structure defined by rho(i,j) = bpd(i) * bpd(j), computes the probability that the vector falls in a rectangle in n-space with error less than eps.

This function calculates the bdp value from sigma, determines the right inf value and calls mvnprd.

---

as251StudentT *Algorithm AS 251: Student T Distribution*

Description

Calculates the Multivariate Normal Distribution with Product Correlation Structure published by Charles Dunnett, Algorithm AS 251.1 Appl.Statist. (1989), Vol.38, No.3, doi:10.2307/2347754.

Usage

```
as251StudentT(
  lower,
  upper,
  sigma,
  ...,
  df,
  eps = 1e-06,
  errorControl = c("strict", "halvingIntervals"),
  intervalSimpsonsRule = 0
)
```

Arguments

lower	Lower limits of integration. Array of N dimensions
upper	Upper limits of integration. Array of N dimensions
sigma	Values defining correlation structure. Array of N dimensions
...	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
df	Degrees of Freedom. Use 0 for infinite D.F.

eps	desired accuracy. Defaults to 1e-06
errorControl	error control. If set to "strict" (default), strict error control based on fourth derivative is used. If set to "halvingIntervals", error control based on halving intervals is used
intervalSimpsonsRule	Interval width for Simpson's rule. Value of zero caused a default .24 to be used

Details

For a multivariate normal vector with correlation structure defined by rho(i,j) = bpd(i) * bpd(j), computes the probability that the vector falls in a rectangle in n-space with error less than eps.

This function calculates the bdp value from sigma, determines the right inf value and calls mvstud.

---

compare *Start or Extend a Trial Comparison*

Description

S3 generic for comparing existing trial planning objects. The ParameterSet method creates a comparison or appends to an existing TrialComparison.

The ParameterSet method starts or extends a trial comparison.

Usage

```
compare(x, y)

## S3 method for class 'ParameterSet'
compare(x, y)
```

Arguments

x	A supported trial planning object or a TrialComparison.
y	A trial design, design characteristics, sample size or power plan, simulation result, list of supported objects, or TrialComparison to add. May be omitted to create a comparison containing only x.

Details

For supported ParameterSet objects, compare(x, y) calls getComparison() with x and y. Supported classes are TrialDesign, TrialDesignCharacteristics, TrialDesignPlan, SimulationResults, PowerAndAverageSampleNumberResult, EventProbabilities, NumberOfSubjects, ConditionalPowerResults (including subclasses), PerformanceScore, FisherInformation, and TrialDesignOptimalConditionalErrorChara
If x is a TrialComparison, it calls x$add(y) instead. The input validation and collection rules of getComparison() apply in both cases. Objects are retained by reference without recalculating them.

Value

A TrialComparison. When x is already a comparison, it is modified by reference if y is supplied. Without y, a new comparison is returned.

See Also

getComparison(), add(), delete(), fetch.TrialComparison(), getObjects()

Examples

```
comp <- getDesignInverseNormal(kMax = 2) |>
    compare(getSampleSizeMeans(alternative = 0.2)) |>
    compare(getSampleSizeMeans(alternative = 0.59))
length(comp)
```

---

delete *Delete an Object from a Collection*

Description

S3 generic for removing objects. The TrialComparison method delegates to $delete() and modifies the existing comparison by reference.

The TrialComparison method deletes one contained object by position or name.

Usage

```
delete(x, ...)

## S3 method for class 'TrialComparison'
delete(x, ...)
```

Arguments

x	An object supporting a delete() method, such as a TrialComparison.
...	For TrialComparison, exactly one index, supplied by position or name: a positive integer position or an exact object name. Invalid selections leave the comparison unchanged. Removing the last object leaves an empty comparison.

Value

The modified comparison for TrialComparison.

See Also

getComparison(), add()

Examples

```
comparison <- getComparison(
    fixed = getDesignFixed(),
    sequential = getDesignGroupSequential(kMax = 2)
)
comparison |> delete("fixed")
comparison |> delete(index = 1)
comparison |>
    add(getDesignFixed()) |>
    delete(1)
```

`fetch.TrialComparison` *Fetch Objects from a Trial Comparison*

Description

Extracts contained objects by name or position without modifying the comparison or copying its objects. A single selection returns the object itself by default.

Usage

```
## S3 method for class 'TrialComparison'
fetch(x, ..., output = c("value", "named", "labeled", "list"))
```

Arguments

`x`	A `TrialComparison`.
`...`	Object names or integer positions, including vectors and multiple selections. Names may be quoted or unquoted. Positive positions count from the beginning and negative positions from the end. With no selection, the last object is returned. Quasi-quotation and programmatic selectors are supported; an unquoted name matching a contained object takes precedence over a variable of the same name in the calling environment.
`output`	One of "value" (default), "named", "labeled", or "list". "value" returns a single selected object directly and multiple selections as a named list. The other modes always return a named list. Collection labels are the supplied object names, with positions for unnamed objects; consequently "labeled" and "named" are equivalent for comparisons.

Details

`fetch(x, "fixed")`, `fetch(x, fixed)` and `fetch(x, 1)` provide the named and positional object access supported by `[[`. Only contained objects are selected; R6 members are never returned as a fallback. Numeric positions must be finite, nonzero integers within the collection bounds. Selecting the last object of an empty comparison raises an index error.

Value

A contained object by reference or a list of objects. An unknown name returns NULL (retained as a NULL entry in list output). Empty selector vectors return an empty list.

See Also

`fetch()` and `getComparison()`.

Examples

```
comparison <- getComparison(
    fixed = getDesignFixed(),
    sequential = getDesignGroupSequential(kMax = 2)
)
identical(comparison |> fetch("fixed"), comparison[["fixed"]])
identical(comparison |> fetch(fixed), comparison$fixed)
```

```
identical(comparison |> fetch(1), comparison[[1]])
comparison |> fetch(-1)
comparison |> fetch(1:2, output = "list")
```

`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 numeric 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

```
## Not run:
# 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

## End(Not run)
```

---

getAnalysisResults *Get Analysis Results*

Description

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

Usage

```
getAnalysisResults(
  design,
  dataInput,
  ...,
  directionUpper = NA,
  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 an element of DatasetGeneral, DatasetMeans, DatasetRates, or 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** stDevH1 **(or** assumedStDev **/** assumedStDevs**),** pi1**,** pi2**, or** piTreatments**,** piContr

The assumed effect size, standard deviation or rates to calculate the conditional power if nPlanned is specified. For survival designs, thetaH1 refers to the hazard ratio. For one-armed trials with binary outcome, only pi1 can be specified, for two-armed trials with binary outcome, pi1 and pi2 can be specified referring to the assumed treatment and control rate, respectively. In multi-armed or enrichment designs, you can specify a value or a vector with elements referring to the treatment arms or the subpopulations, respectively. For testing rates, the parameters to be specified are piTreatments and piControl (multi-arm designs) and piTreatments and piControls (enrichment designs).
If not specified, the conditional power is calculated under the assumption of observed effect sizes, standard deviations, rates, or hazard ratios.

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

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`.

`stdErrorEstimate` Estimate of standard error for calculation of final confidence intervals for comparing rates in two treatment groups, default is `"pooled"`.

`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` Logical. Specifies the direction of the alternative, only applicable for one-sided testing; `TRUE` means that larger values of the test statistics yield smaller p-values. The default is `TRUE` for means, rates, and count data, and `FALSE` for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified `directionUpper` must agree with it.

`thetaH0` The null hypothesis value, default is `0` for general estimates, the normal case, and the binary case, 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

- *general estimates*: a value on the scale of the supplied estimates can be specified.
- *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.
- *count data*: a bound for testing H0: `lambda1 / lambda2 = 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 subpopulation.

`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. For simulating means and rates for a two treatment groups design, it can be a vector of length `kMax`, the number of stages. It can be a vector of length `kMax`, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally `allocationRatioPlanned` is treated as a vector of length `kMax`, not a scalar.

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

`maxInformation`
: Positive 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 if the planned sample size for the subsequent stages (`nPlanned`) is specified. The conditional power is calculated either under the assumption of the observed effect or under the assumption of an assumed effect, that has to be specified (see above).
For testing rates in a two-armed trial, pi1 and pi2 typically refer to the rates in the treatment and the control group, respectively. This is not mandatory, however, and so pi1 and pi2 can be interchanged. In many-to-one multi-armed trials, piTreatments and piControl refer to the rates in the treatment arms and the one control arm, and so they cannot be interchanged. piTreatments and piControls in enrichment designs can principally be interchanged, but we use the plural form to indicate that the rates can be differently specified for the sub-populations.

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.

Note

Repeated p-values are calculated by identifying the smallest global significance level for which the observed test statistic would lead to rejection under the same design rule. Therefore, they require that the design can be re-created for arbitrary values of the global significance level. For user-defined designs with fixed userAlphaSpending values, this alpha-dependent design family is not uniquely defined. Consequently, repeated p-values are not available for such designs. Repeated p-values are also unavailable for sequential designs with typeBetaSpending = "bsUser", regardless of whether futility is binding. These calculations return NA with a warning; other analysis results remain available.

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

getObservedInformationRates()

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

Examples

```
## Not run:
# 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 subpopulation (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)

## End(Not run)
```

`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

```
## Not run:
# 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)

## End(Not run)
```

`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 `getDesignConditionalDunn`
See Koenig et al. (2008) and Wassmer & Brannath (2025), 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

```
## Not run:
# 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)

## End(Not run)
```

---

getComparison *Compare Existing Trial Planning Objects*

Description

Creates a comparison of existing designs, design characteristics, sample size or power plans, simulation results, and power/ASN results without recalculating them. A single comparison summary returns its underlying comparison.

`length()` returns the number of objects in the comparison.

`names()` returns the supplied names of the comparison objects.

Usage

```
getComparison(..., objects = NULL)

## S3 method for class 'TrialComparison'
length(x)

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

Arguments

`...`	Designs, design characteristics, plans, simulations, `PowerAndAverageSampleNumberResult`, `EventProbabilities`, `NumberOfSubjects`, `ConditionalPowerResults` (including subclasses), `PerformanceScore`, `FisherInformation`, or `TrialDesignOptimalCondition` objects, lists thereof, `TrialComparison` objects, or `TrialComparisonSummary` objects. Entries from `...` precede entries from `objects`.
`objects`	Optional objects, lists of objects, `TrialComparison` objects, or `TrialComparisonSummary` objects. NULL is treated as an empty list.
`x`	A `TrialComparison`.

Details

The comparison retains references to its objects. Changes to an original object are reflected in subsequent comparisons; no comparison data are cached. `$getObjects()` and `getObjects()` return the named list of references. Replacing entries in that returned list does not change the comparison. `length()` is the number of objects and `names()` returns their supplied names (or `NULL`). Retrieve contained objects using `comparison[[i]]`, `comparison[["name"]]` or `comparison$name`. R6 members take precedence for `$`; use `[[` to retrieve an object whose name coincides with a member such as add. Unknown names return NULL. The `fetch.TrialComparison()` method provides the same object access through `fetch(comparison, name)` or `fetch(comparison, position)`. A single summary can be unwrapped with `summaryObject |> getComparison()`, returning exactly `summaryObject$object`. Changes to the returned collection therefore affect that underlying comparison. In combinations or nested lists, summaries contribute their underlying comparisons' current objects; a new collection is created without changing the source collections. Summary display settings are not transferred to the comparison.

Lists and comparisons are recursively expanded in their original order. Names on individual objects are retained; container names are discarded. Supplied non-empty names must be unique across the expanded collection; duplicate names raise an error. Unnamed entries and repeated references to the same object are allowed. getComparison() creates an empty comparison. Its data frame has zero rows and columns, and plotting requires numeric data to be added first.

`comparison$add(..., objects = NULL)` appends objects with the same input rules and returns the comparison invisibly, supporting chaining. All new entries are validated before the stored list is changed, including name collisions with existing objects. Adding another comparison copies its current list of references; subsequent additions to that source comparison do not affect the target. Existing summary objects remain snapshots; call `summary()` again after changing the collection.

`comparison$delete(index)` removes one object by its positive integer position or exact name and returns the comparison invisibly. Invalid positions or unknown names raise an error without

changing the collection. Removing the last object leaves an empty comparison, which can be filled again. Names and order of remaining objects are retained; positions are renumbered. The S3 generics `add()` and `delete()` delegate to these methods and support pipelines such as `comparison |> add(other) |> delete(1)`. Shallow cloning shares the contained objects. `clone(deep = TRUE)` is unsupported.

`print()` describes the collection and selected observed ranges. `summary()` groups differences by object scope, inputs/design settings and results, with equal profiles grouped by object number. Parameter provenance and native dimensions determine the applicable profile coordinates. Repeated design settings on arm/category rows are removed after exact consistency checks. Genuine scalar parameters are compared independently of the number of rows; dimensioned parameters retain their stage, group and scenario domains even when their values happen to be equal. Constant numeric vector cells, such as `1, 1`, are normalized to the corresponding scalar value. Missing values and their positions participate in exact comparisons. No rounding or numerical tolerance is used to decide whether profiles differ.

For comparisons containing interim analyses, fixed plans' `nFixed`, `nFixed1`, and `nFixed2` are harmonized to `numberOfSubjects`, `numberOfSubjects1`, and `numberOfSubjects2`. The corresponding stored group sizes are included for sequential plans. Comparisons of fixed plans alone retain the fixed names. With `includeAllParameters = TRUE`, sequential plans' `nFixed` reference values remain separate from their stage-wise sample sizes. Fixed survival plans also use `cumulativeEventsPerStage` for `eventsFixed` when interim analyses occur in the comparison; sequential plans retain their fixed-design event reference.

Available sample-size expectations `expectedNumberOfSubjectsH1` and `expectedEventsH1` use `expectedNumberOfSubjects` and `expectedNumberOfEvents`. These expectations refer to each scenario's supplied alternative. Harmonization does not change alternatives or stopping rules, combine H0/H0-H1 expectations, or reinterpret simulation assumptions used for conditional power. Conflicting non-missing target values remain in separate columns. Power/ASN results from `getPowerAndAverageSampl` retain their generic `theta`, `nMax`, and `averageSampleNumber` scales. Individual `theta` scenarios retain their stage coordinates even in full exports. The comparison includes `nMax` even when the compact source data frame omits it; no endpoint or conversion to subjects or events is inferred. With column harmonization enabled, their stage-wise `earlyStop` becomes `earlyStopPerStage`, `overallEarlyStop` becomes `earlyStop`, and `overallFutility` becomes `futilityStop`. `calculatedPower` merges into `overallReject` when the values agree. Original names remain available with `harmonizeColumnNames = FALSE` in data frames and summaries. Original objects are unchanged.

Event probabilities and recruitment results retain their time coordinates and assumptions. Conditional-power results retain planned stages, treatment arms or populations, and observed stage and data context; no analysis is rerun. Performance scores retain scenario alternatives, simulation inputs and design settings. Fisher information retains selected stages, situations, the source plan when available, and its cumulative or stage-wise type; these scales are not harmonized. Profiles and plot groups respect time, arm, population and situation coordinates.

Console and Markdown output are bounded. Limits and display rounding do not change the underlying data. `as.data.frame.TrialComparison()` returns the full comparison table. `plot.TrialComparison()` selects axes only when unambiguous. `getObjectRCode()` reconstructs the comparison and its contained objects; executing generated simulation code reruns the corresponding simulations.

Value

An R6 `TrialComparison` inheriting from `ParameterSet`. For a single `TrialComparisonSummary`, its underlying comparison is returned by reference; otherwise a new collection is created.

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

- add() to obtain the field names,
- delete() to obtain the field names,
- fetch() to obtain the field names,
- compare() to obtain the field names,
- names() to obtain the field names,
- length() 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

compare(), add(), delete(), fetch.TrialComparison(), getObjects()

Examples

```
asn <- getPowerAndAverageSampleNumber(getDesignGroupSequential(kMax = 2),
    theta = c(0, 0.2), nMax = 100
)
getComparison(asn, asn)

comparison <- getComparison(list(
    fixed = getDesignFixed(), sequential = getDesignGroupSequential(kMax = 2)
))
print(comparison)
comparisonSummary <- summary(comparison)
identical(comparisonSummary |> getComparison(), comparison)
head(as.data.frame(comparison))
withCharacteristics <- getComparison(
    getDesignGroupSequential(kMax = 2),
    getDesignCharacteristics(getDesignGroupSequential(kMax = 3))
)
combined <- getComparison(comparison, other = getDesignFixed(alpha = 0.05))
combined$add(list(getDesignFixed(alpha = 0.01)))
empty <- getComparison()
empty$add(getDesignFixed())
combined$delete(1)
combined$delete("sequential")
## Not run:
plot(comparison,
    xAxis = "objectNumber",
    yAxis = "criticalValues",
```

```
    plotLineEnabled = FALSE
)

## End(Not run)
```

`getComparisonParameterDescriptions` *Describe Comparison Parameters*

Description

Formats comparison parameter metadata for console output, summaries, and UI labels. Formatting does not alter the metadata or comparison data.

Usage

```
getComparisonParameterDescriptions(
  parameterInfo,
  ...,
  digits = NA_integer_,
  showDetails = TRUE
)
```

Arguments

`parameterInfo`	A comparison parameter metadata table, as available in `summary(comparison)$parameterInfo`, or a row subset of such a table.
`...`	Further arguments. Currently unused.
`digits`	Optional number of decimal places for displayed values.
`showDetails`	Whether to include the inferred type and plotting role.

Value

A character vector named by parameter names. Empty input returns character(). Empty columns are described as empty, not as single-valued.

Examples

```
comparison <- getComparison(getDesignFixed(), getDesignGroupSequential(kMax = 2))
info <- summary(comparison)$parameterInfo
getComparisonParameterDescriptions(info)
```

`getComparisonParameterNames` *Select Comparison Parameters*

Description

Selects varying parameters, axis candidates, or grouping variables from comparison-data columns. Column order is retained.

Usage

```
getComparisonParameterNames(
  data,
  ...,
  varyingOnly = TRUE,
  type = "all",
  exclude = character(),
  maxNumberOfLevels = 5L,
  binaryIntegerAsFactor = TRUE
)
```

Arguments

`data`	A data.frame, typically obtained with as.data.frame() from a trial comparison. Columns must be one-dimensional atomic vectors, including factors and dates.
`...`	Further arguments. Currently unused.
`varyingOnly`	Whether to exclude constant and empty columns.
`type`	One of "all", "continuous", "factor", "binary", or "groupable". Groupable columns are categorical or have at most maxNumberOfLevels observed values. An empty column is never groupable. Continuous binary columns remain eligible as continuous axes.
`exclude`	Character vector of names to exclude, e.g., identifiers or variables already selected for an axis or a category.
`maxNumberOfLevels`	Positive upper limit for numerical grouping variables.
`binaryIntegerAsFactor`	Whether integer-valued binary variables are also eligible for use as factors. This does not change the source column.

Value

A character vector of column names. No plot or GUI state is changed.

Examples

```
data <- data.frame(alpha = 0.025, effect = c(0.2, 0.3), power = c(0.8, 0.9))
getComparisonParameterNames(data)
getComparisonParameterNames(data, exclude = "effect", type = "continuous")
```

`getComparisonPlotParameters` *Select Parameters for a Comparison Plot*

Description

Proposes axes and an object category without creating a plot. Selection problems are returned as text so applications can display them before plotting. The four column selectors accept technical names, dictionary captions, and object-specific captions, as in `fetch()`. Unbound names may be supplied without quotes; defined variables and expressions are evaluated normally.

Usage

```
getComparisonPlotParameters(
  data,
  xAxis = NULL,
  yAxis = NULL,
  ...,
  category = NULL,
  group = NULL,
  objects = NULL,
  yAxisSide = "auto"
)
```

Arguments

`data`	A comparison data frame, a `TrialComparison`, or its summary.
`xAxis, yAxis`	Optional explicit axis names or captions. `yAxis` may be a vector of distinct numeric columns. Automatic axes must be varying numeric columns. Every y column must be numeric; an explicitly selected x-axis may also be categorical.
`...`	Further arguments. Currently unused.
`category`	Optional character vector of category column names. All selected columns jointly define the categories. NULL selects `objectName`, `objectNumber`, or `designNumber`, in that order, when it identifies multiple objects and is not selected as an axis or group. NA_character_ disables automatic categories, as does character(0). No category is needed for a single object.
`group`	Optional explicit grouping column.
`objects`	Optional original objects for role-aware selection when `data` is a data frame. A `TrialComparison` supplies its own objects automatically.
`yAxisSide`	Axis assignment: "auto" (default), "left", or "right". Supply one value for all y parameters, a vector in their order, or a named vector assigning individual parameters. Unspecified named entries remain automatic. The same argument is accepted by `plot.TrialComparison()`.

Details

With original objects available, comparisons in which each object contributes one point at a distinct finite numeric x value are connected across objects when only that input varies and the object scope

is shared. Automatic object categories are omitted in this case and `connectAcrossObjects` is TRUE. Explicit categories or groups retain their separate curves.

Axis-selection messages use a short problem statement, separate candidate lists with names and captions, and one example using eligible axes. Text wraps to the console width (up to 100 characters). Candidate lists show at most five entries per axis and may be shortened further to keep the message compact. Full candidates and parameter descriptions remain available in `issues`.

Explicit selections prevail. When original objects are available, inputs and design settings are preferred for x and results for y, as in the comparison plot method. Without original objects, only varying numeric columns determine automatic axes. Identifiers are excluded, but stages remain eligible. An omitted axis is selected only when exactly one eligible candidate remains. Ambiguity, invalid column selections, and unsupported data structures are returned without raising a selection error. A category may remain NULL without an error when no object category is available or required. The data and the original objects are not changed, and no ggplot2 installation is required.

Automatic y-axis assignment uses the largest gap between finite nonzero parameter magnitudes (maximum absolute values). A gap of at least a factor of ten separates the smaller scale onto the right axis. Otherwise all parameters share the left axis. Consistent explicit reversals also reverse automatic assignments; explicit assignments always prevail. There are at most two shared scales, with no separate normalization within either side. Dual-axis reference ranges include zero. Probability-like ranges within numerical tolerance of `[0, 1]`, with maximum at least 0.1, use `[0, 1]`. Smaller magnitudes retain their observed range. Constant zero or entirely missing ranges use `[0, 1]` to avoid a degenerate transformation.

Value

A list containing `xAxis`, `yAxis`, `category`, `group`, `connectAcrossObjects`, `yAxisSide`, `yAxisTransformation`, `yAxisRanges`, `message`, `candidates`, and `issues`. `yAxisSide` is named by the selected parameters; the right-axis plotting transformation is `value * scale + offset` from `yAxisTransformation`. `yAxisRanges` contains the reference ranges used for scaling, not ggplot's expanded panel limits. Unresolved selections are NULL. `message` is NULL on success, otherwise summarizes selection problems. `candidates` lists eligible columns for each selection; `issues` contains diagnostic metadata used by `plot.TrialComparison()`.

Examples

```
comparison <- getComparison(
    first = getDesignFixed(alpha = 0.025),
    second = getDesignFixed(alpha = 0.05)
)
selection <- getComparisonPlotParameters(comparison, yAxis = "criticalValues")
selection[c("xAxis", "yAxis", "category", "message")]
```

---

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** `stDevH1` **(or** `assumedStDev` **/** `assumedStDevs`**),** `pi1`**,** `pi2`**, or** `piTreatments`**,** `piContr`
: The assumed effect size, standard deviation or rates to calculate the conditional power if `nPlanned` is specified. For survival designs, `thetaH1` refers to the hazard ratio. For one-armed trials with binary outcome, only `pi1` can be specified, for two-armed trials with binary outcome, `pi1` and `pi2` can be specified referring to the assumed treatment and control rate, respectively. In multi-armed or enrichment designs, you can specify a value or a vector with elements referring to the treatment arms or the sub-populations, respectively. For testing rates, the parameters to be specified are `piTreatments` and `piControl` (multi-arm designs) and `piTreatments` and `piControls` (enrichment designs).
If not specified, the conditional power is calculated under the assumption of observed effect sizes, standard deviations, rates, or hazard ratios.

`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 conditional 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 subpopulation.

`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. For simulating means and rates for a two treatment groups design, it can be a vector of length `kMax`, the number of stages. It can be a vector of length `kMax`, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally `allocationRatioPlanned` is treated as a vector of length `kMax`, not a scalar.

Details

The conditional power is calculated if the planned sample size for the subsequent stages is specified. For testing rates in a two-armed trial, pi1 and pi2 typically refer to the rates in the treatment and the control group, respectively. This is not mandatory, however, and so pi1 and pi2 can be interchanged. In many-to-one multi-armed trials, piTreatments and piControl refer to the rates in the treatment arms and the one control arm, and so they cannot be interchanged. piTreatments and piControls in enrichment designs can principally be interchanged, but we use the plural form to indicate that the rates can be differently specified for the sub-populations.

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(), getClosedConditionalD
getConditionalRejectionProbabilities(), getFinalConfidenceInterval(), getFinalPValue(),
getRepeatedConfidenceIntervals(), getRepeatedPValues(), getStageResults(), getTestActions()

Examples

```
## Not run:
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
)

## End(Not run)
```

`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()`, `getClosedConditionalD`
`getConditionalPower()`, `getFinalConfidenceInterval()`, `getFinalPValue()`, `getRepeatedConfidenceInterva`
`getRepeatedPValues()`, `getStageResults()`, `getTestActions()`

Examples

```
## Not run:
# Calculate CRP for a Fisher's combination test design with
# two remaining stages and check the results by simulation.
design <- getDesignFisher(
    kMax = 4, alpha = 0.01,
    informationRates = c(0.1, 0.3, 0.8, 1)
)
```

```
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
)

## End(Not run)
```

`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()`. Multi-arm simulation data additionally contain `selectedForNextStage`. At non-final stages, this logical value indicates whether the adaptation selected the arm to continue, conditional on the trial not stopping; it is `NA` at the final stage. In contrast, `rejectPerStage` indicates rejection of the arm's null hypothesis, while `futilityPerStage` is a trial-level futility-stopping decision repeated for the arm rows of the stage. For piecewise multi-arm survival simulations, `scenario` identifies the situation by its one-based position in the third dimension of `piecewiseSurvivalTime$hazardRatio`. It matches `scenario` in `getRawData()` and remains distinct when situations have identical first-interval effects. The `effect` and `omegaMax` columns describe the first interval only.

Value

Returns a `data.frame`.

Examples

```
## Not run:
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)

## End(Not run)
```

---

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 DatasetGeneral, DatasetMeans, DatasetRates, or DatasetSurvival can be created as follows:

- An element of DatasetGeneral for general estimates is created by getDataset(est =, se =, df =) where est, se, and optionally df are vectors with stage-wise estimates, standard errors, and degrees of freedom. If df is omitted, the degrees of freedom are set to Inf, corresponding to a normal approximation.
- 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 (subpopulation wise) or stratified data input.
For non-stratified (subpopulation 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 can only 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 cumEvents, cumExpectedEvents, cumVarianceEvents, and cumAllocationRatios or overallEvents, overallExpectedEvents, overallVarianceEvents, and overallAllocationRatios. From this, (stratified) log-rank tests and and the independent increments 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

```
## Not run:
# Create a Dataset of General Estimates without specifying degrees
# of freedom (internally, df is set to Inf for all stages):
datasetOfGeneralEstimates <- getDataset(
    est = c(0.10, 0.25, 0.38),
    se  = c(0.20, 0.18, 0.15)
)
datasetOfGeneralEstimates
datasetOfGeneralEstimates$degreesOfFreedom

# It is also possible to specify the degrees of freedom for each stage.
```

```
# Typically these come from a linear model analysis, e.g. with repeated measures
# using the [`mmrm`](https://cran.r-project.org/package=mmrm) package.
datasetOfLeastSquareMeans <- getDataset(
    est = c(0.10, 0.25, 0.38),
    se  = c(0.20, 0.18, 0.15),
    df  = c(10.3, 12.5, 15.1)
)
datasetOfLeastSquareMeans

# 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)

datasetOfMeans2 <- 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)
)
datasetOfMeans2
datasetOfMeans2$show(showType = 2)
as.data.frame(datasetOfMeans2)

# Create a Dataset of Means (two groups):
datasetOfMeans3 <- 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)
)
datasetOfMeans3

datasetOfMeans4 <- 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)
)
datasetOfMeans4

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)
)
```

```
datasetOfMeans5 <- getDataset(df)
datasetOfMeans5

# Create a Dataset of Means (three groups) where the comparison of
# treatment arm 1 to control is dropped at the second interim stage:
datasetOfMeans6 <- 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))
datasetOfMeans6

# 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):
datasetOfRates2 <- 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)
)
datasetOfRates2

# Create a Dataset of Rates (three groups) where the comparison of
# treatment arm 2 to control is dropped at the first interim stage:
datasetOfRates3 <- 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))
datasetOfRates3

# 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.
datasetSurvival2 <- 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)
)
datasetSurvival2

# 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

## End(Not run)
```

`getDesignCharacteristics` *Get Design Characteristics*

Description

Calculates the characteristics of a design and returns it.

Usage

```
getDesignCharacteristics(design = NULL, ...)
```

Arguments

`design`	A design object. If omitted, a default design is created.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.

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.

For an optimal conditional error design, supply `theta` through `...` as a finite numeric vector of treatment effects on the mean difference scale. The design is evaluated at those effects without recalibration.

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.

References

Brannath, W. & Bauer, P. (2004). Optimal conditional error functions for the control of conditional power. Biometrics, 60(3), 715-723. doi:10.1111/j.0006341X.2004.00221.x

Brannath, W., zur Verth, J., Dreher, M. & Scharpenberg, M. (2024; revised 2026). Optimal monotone conditional error functions. doi:10.48550/arXiv.2402.00814

The related R package optconerrf: Optimal Monotone Conditional Error Functions provides a standalone implementation of this methodology. The rpact implementation uses rpact design objects and interfaces; optconerrf is not required to run these functions.

See Also

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

Examples

```
## Not run:
# 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"))

## End(Not run)
```

---

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,
  directionUpper = NA
)
```

Arguments

`alpha`	The significance level alpha, default is `0.025`. Must be a positive numeric of length 1.
`informationAtInterim`	The information to be expected at interim, default is `informationAtInterim = 0.5`.
`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`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).
`directionUpper`	Logical. Specifies the direction of the alternative, only applicable for one-sided testing; `TRUE` means that larger values of the test statistics yield smaller p-values. The default is `TRUE` for means, rates, and count data, and `FALSE` for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified `directionUpper` must agree with it.

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 (2025), 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(), getDesignFixed(), 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_,
  alpha0Scale = c("pValue", "zValue", "condPowerAtObserved", "predictivePower"),
  informationRates = NA_real_,
  sided = 1,
  bindingFutility = NA,
  directionUpper = 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. Must be a positive integer of length 1 (default value 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. Must be a positive numeric of length 1.
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.

alpha0Scale
: Character. The scale of the futility bounds. Must be one of "pValue", "zValue", "condPowerAtObserved", or "predictivePower". Default is "pValue".

informationRates
: The information rates t_1, ..., t_kMax (that must be fixed prior to the trial), default is (1:kMax) / kMax. For the weighted inverse normal design, the weights are derived through w_1 = sqrt(t_1), and w_k = sqrt(t_k - t_(k-1)). For the weighted Fisher's combination test, the weights (scales) are w_k = sqrt((t_k - t_(k-1)) / t_1) (see the documentation).

sided
: Is the alternative one-sided (1) or two-sided (2), default is 1. Must be a positive integer of length 1.

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

directionUpper
: Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.

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

getComparison() to compare different design situations.

getFutilityBounds() for the specification of futility bounds on scales other than the p-value scale.

Vignette: Enhanced Futility Bounds Specification

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

Examples

```
## Not run:
# 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)

## End(Not run)
```

---

getDesignFixed *Get Fixed Design*

Description

Create a fixed (single-stage) trial design. This convenience wrapper constructs an object of class TrialDesignFixed with kMax = 1.

Usage

```
getDesignFixed(
  alpha = NA_real_,
  beta = NA_real_,
  sided = 1L,
  directionUpper = NA,
  twoSidedPower = NA
)
```

Arguments

alpha	The significance level alpha, default is 0.025. Must be a positive numeric of length 1.

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. Must be a positive numeric of length 1.
sided	Is the alternative one-sided (1) or two-sided (2), default is 1. Must be a positive integer of length 1.
directionUpper	Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.
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.#'

Details

The fixed design represents a single-stage hypothesis test. The returned object can be used with the standard analysis and plotting helpers in the package. Typical use is to specify the type I error alpha and the desired type II error beta (or equivalently power = 1 - beta).

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.

See Also

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

Examples

```
# Basic fixed design with default alpha and beta
design <- getDesignFixed()
design

# Custom significance and power
design2 <- getDesignFixed(alpha = 0.05, beta = 0.1)
design2
```

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_,
  efficacyStops = NA,
  futilityStops = NA,
  gammaB = NA_real_,
  bindingFutility = NA,
  futilityBoundsScale = c("zValue", "pValue", "reverseCondPower", "condPowerAtObserved",
    "predictivePower"),
  directionUpper = 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. Must be a positive integer of length 1 (default value 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. Must be a positive numeric of length 1.

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. Must be a positive numeric of length 1.

sided — Is the alternative one-sided (1) or two-sided (2), default is 1. Must be a positive integer of length 1.

informationRates — The information rates t_1, ..., t_kMax (that must be fixed prior to the trial), default is (1:kMax) / kMax. For the weighted inverse normal design, the weights are derived through w_1 = sqrt(t_1), and w_k = sqrt(t_k - t_(k-1)). For the weighted Fisher's combination test, the weights (scales) are w_k = sqrt((t_k - t_(k-1)) / t_1) (see the documentation).

futilityBounds — The futility bounds, defined on the scale defined by futilityBoundsScale. (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. Required for typeOfDesign = "WT".

deltaPT1 — Delta1 for Pampallona & Tsiatis class rejecting H0 boundaries. Required (together with deltaPT0) for typeOfDesign = "PT".

deltaPT0 — Delta0 for Pampallona & Tsiatis class rejecting H1 boundaries. Required (together with deltaPT1) for typeOfDesign = "PT".

optimizationCriterion — Optimization criterion for optimum design within Wang & Tsiatis class ("ASNH1", "ASNIFH1", "ASNsum"), default is "ASNH1", see details.

gammaA — Parameter for alpha spending function. Required for typeOfDesign = "asKD" (validated range [0.4, 8]) and typeOfDesign = "asHSD".

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.

efficacyStops — Logical vector of length kMax - 1 indicating efficacy stops. Default is NA.

futilityStops — Logical vector of length kMax - 1 indicating futility stops. Default is NA.

gammaB — Parameter for beta spending function. Required for typeBetaSpending = "bsKD" (validated range [0.4, 8]) and typeBetaSpending = "bsHSD".

bindingFutility — Logical. 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).

`futilityBoundsScale`
: Character. The scale of the futility bounds. Must be one of "zValue", "pValue", "reverseCondPower", "condPowerAtObserved", or "predictivePower". Default is "zValue".

`directionUpper`
: Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.

`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.

getFutilityBounds() for the specification of futility bounds on scales other than the z-value scale.

Vignette: Enhanced Futility Bounds Specification

Other design functions: getDesignCharacteristics(), getDesignConditionalDunnett(), getDesignFisher(), getDesignFixed(), getDesignInverseNormal(), getGroupSequentialProbabilities(), getPowerAndAverageSa

Examples

```
## Not run:
# 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 70% of the maximum information was observed (default alpha = 0.025)
getDesignGroupSequential(informationRates = c(0.4, 0.7), typeOfDesign = "asP")

## End(Not run)
```

---

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_,
  efficacyStops = NA,
  futilityStops = NA,
  gammaB = NA_real_,
  bindingFutility = NA,
  futilityBoundsScale = c("zValue", "pValue", "reverseCondPower", "condPowerAtObserved",
    "predictivePower"),
  directionUpper = 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`. Must be a positive integer of length 1 (default value 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`. Must be a positive numeric of length 1.
`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`. Must be a positive numeric of length 1.
`sided`	Is the alternative one-sided (`1`) or two-sided (`2`), default is `1`. Must be a positive integer of length 1.
`informationRates`	The information rates t_1, ..., t_kMax (that must be fixed prior to the trial), default is `(1:kMax) / kMax`. For the weighted inverse normal design, the weights are derived through w_1 = sqrt(t_1), and w_k = sqrt(t_k - t_(k-1)). For the

weighted Fisher's combination test, the weights (scales) are w_k = sqrt((t_k - t_(k-1)) / t_1) (see the documentation).

`futilityBounds` The futility bounds, defined on the scale defined by `futilityBoundsScale`. (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. Required for `typeOfDesign = "WT"`.

`deltaPT1` Delta1 for Pampallona & Tsiatis class rejecting H0 boundaries. Required (together with `deltaPT0`) for `typeOfDesign = "PT"`.

`deltaPT0` Delta0 for Pampallona & Tsiatis class rejecting H1 boundaries. Required (together with `deltaPT1`) for `typeOfDesign = "PT"`.

`optimizationCriterion`
Optimization criterion for optimum design within Wang & Tsiatis class (`"ASNH1"`, `"ASNIFH1"`, `"ASNsum"`), default is `"ASNH1"`, see details.

`gammaA` Parameter for alpha spending function. Required for `typeOfDesign = "asKD"` (validated range `[0.4, 8]`) and `typeOfDesign = "asHSD"`.

`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.

`efficacyStops` Logical vector of length `kMax - 1` indicating efficacy stops. Default is `NA`.

`futilityStops` Logical vector of length `kMax - 1` indicating futility stops. Default is `NA`.

`gammaB` Parameter for beta spending function. Required for `typeBetaSpending = "bsKD"` (validated range `[0.4, 8]`) and `typeBetaSpending = "bsHSD"`.

`bindingFutility`
Logical. 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`).

`futilityBoundsScale`
Character. The scale of the futility bounds. Must be one of `"zValue"`, `"pValue"`, `"reverseCondPower"`, `"condPowerAtObserved"`, or `"predictivePower"`. Default is `"zValue"`.

`directionUpper` Logical. Specifies the direction of the alternative, only applicable for one-sided testing; `TRUE` means that larger values of the test statistics yield smaller p-values.

The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.

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(), getDesignFixed(), getDesignGroupSequential(), getGroupSequentialProbabilities(), getPowerAndAverage

Examples

```
## Not run:
# 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)

## End(Not run)
```

---

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`.

Deprecated

Use `getComparison()` instead.

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

```
## Not run:
# 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)

## End(Not run)
```

`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 numeric 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(q = time, shape = kappa, scale = 1 / lambda) of the stats package.
For example, getPiecewiseExponentialDistribution(time = 130, piecewiseLambda = 0.01, kappa = 4.2) and pweibull(q = 130, shape = 4.2, scale = 1 / 0.01) provide the same 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). Must be a positive numeric of length 1, or a vector with one hazard rate per time interval if piecewiseSurvivalTime is specified.

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). Must be a positive numeric value or vector: without piecewiseSurvivalTime, a vector defines alternative hazard rates to be considered; with piecewiseSurvivalTime, it contains one hazard rate per time interval. getEventProbabilities() uses only the first value of a non-piecewise vector.

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. Must be a positive numeric value or vector; getEventProbabilities() accepts a single value only.

dropoutRate1
: The assumed drop-out rate in the treatment group, default is 0. For multi-arm simulations, this can be a vector of length equal to the number of active arms, specifying different drop-out rates for each active arm.

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

```
## Not run:
# 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')

## End(Not run)
```

---

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 = NA,
  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 an element of `DatasetGeneral`, `DatasetMeans`, `DatasetRates`, or `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`.

`stdErrorEstimate` Estimate of standard error for calculation of final confidence intervals for comparing rates in two treatment groups, default is `"pooled"`.

`directionUpper` Logical. Specifies the direction of the alternative, only applicable for one-sided testing; `TRUE` means that larger values of the test statistics yield smaller p-values. The default is `TRUE` for means, rates, and count data, and `FALSE` for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified `directionUpper` must agree with it.

`thetaH0` The null hypothesis value, default is `0` for general estimates, the normal case, and the binary case, 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

- *general estimates*: a value on the scale of the supplied estimates can be specified.
- *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.
- *count data*: a bound for testing H0: `lambda1 / lambda2 = 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`. Must be a positive numeric of length 1.

`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()`, `getClosedConditionalD`
`getConditionalPower()`, `getConditionalRejectionProbabilities()`, `getFinalPValue()`, `getRepeatedConfide`
`getRepeatedPValues()`, `getStageResults()`, `getTestActions()`

Examples

```
## Not run:
design <- getDesignInverseNormal(kMax = 2)
data <- getDataset(
    n = c(20, 30),
    means = c(50, 51),
    stDevs = c(130, 140)
)
getFinalConfidenceInterval(design, dataInput = data)

## End(Not run)
```

---

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(), getClosedConditionalD
getConditionalPower(), getConditionalRejectionProbabilities(), getFinalConfidenceInterval(),
getRepeatedConfidenceIntervals(), getRepeatedPValues(), getStageResults(), getTestActions()

Examples

```
## Not run:
design <- getDesignInverseNormal(kMax = 2)
data <- getDataset(
    n      = c( 20,  30),
    means  = c( 50,  51),
    stDevs = c(130, 140)
)
getFinalPValue(getStageResults(design, dataInput = data))

## End(Not run)
```

---

getFisherInformation *Get Fisher Information From a Design Plan or Simulation Results*

Description

Calculates cumulative or stage-wise Fisher information at planned analyses for a design plan or simulation results object for means, rates, survival, or count data endpoints. This is particularly useful for calculating the information argument of getFutilityBounds().

Usage

```
getFisherInformation(
  designPlan,
  stage = NA_integer_,
  type = c("cumulative", "stageWise")
)
```

Arguments

`designPlan`	A trial design plan or simulation results object as returned by functions such as `getSampleSizeMeans()`, `getPowerMeans()`, `getSampleSizeRates()`, `getPowerRates()`, `getSampleSizeSurvival()`, `getPowerSurvival()`, `getSampleSizeCounts()`, `getPowerCounts()`, `getSimulationMeans()`, `getSimulationRates()`, `getSimulationSurvival(`, `getSimulationCounts()`, or the corresponding multi-arm simulation functions.
`stage`	Integer vector. The analysis stage or stages for which Fisher information is requested. If `NA` (default), all stages of the design are used.
`type`	Character. Defines whether cumulative information through each requested analysis (`"cumulative"`, the default) or the information increment contributed by each requested stage (`"stageWise"`) is returned.

Details

Cumulative information

With `type = "cumulative"`, the function returns the total Fisher information available at each requested analysis, including information accumulated during all preceding stages. With `type = "stageWise"`, it returns the increment contributed by each requested stage. The stage-wise value for stage `j > 1` is calculated by subtracting the cumulative information at stage `j - 1` from that at stage `j`, even if stage `j - 1` was not included in `stage`. At the first analysis, cumulative and stage-wise information are identical.

To obtain stage-wise increments from cumulative values $I_1, \ldots, I_k$, use $I_1, I_2 - I_1, \ldots, I_k - I_{k-1}$. For example:

```
informationCumulative <- as.numeric(getFisherInformation(
    designPlan,
    stage = 1:2
))
informationStageWise <- c(
    informationCumulative[1],
    diff(informationCumulative)
)
```

If the result contains several planning alternatives or treatment comparisons, apply the differences separately to each corresponding series. The function performs this calculation directly when `type = "stageWise"`; the explicit calculation above illustrates its definition.

Calculation by endpoint

Means For a one-sample comparison, information is the cumulative sample size divided by the variance. For a two-sample comparison, it is the inverse variance of the estimated treatment difference, based on cumulative group sizes, standard deviations, and the planned allocation ratio. For a mean-ratio analysis, the null value is additionally taken into account.

Rates Information is the inverse binomial variance of the estimated rate or rate difference under the planning assumptions. It uses cumulative planned sample sizes, event probabilities, and, for multiple groups, the planned allocation ratio.

Survival Information is based on the cumulative number of events at the requested analysis and the planned allocation ratio. For multi-arm designs, the available events are apportioned to the relevant treatment comparisons using the planning assumptions.

Count data Information is based on the negative binomial model and includes all exposure accumulated up to the requested analysis time. It accounts for recruitment, exposure or follow-up time, event rates, allocation ratio, and overdispersion. If the design plan already contains information by analysis, those stored cumulative values are used.

Use with getFutilityBounds

For a conversion involving `"effectEstimate"`, pass the cumulative information returned for the relevant analysis as `information[1]`. For conditional- or predictive-power conversions in a two-stage setting, `getFutilityBounds()` interprets `information[1]` as the cumulative information at the first analysis and `information[2]` as the additional information after that analysis. It can therefore be populated directly with:

```
information <- getFisherInformation(
    designPlan,
    stage = 1:2,
    type = "stageWise"
)
```

For an effect-estimate conversion at analysis `j`, use:

```
information <- getFisherInformation(
    designPlan,
    stage = j,
    type = "cumulative"
)
```

The `type` field in every result enables `getFutilityBounds()` to detect incompatible use. The complete result can also be piped into `getFutilityBounds()`. In that case, its `stage` and `situations` fields are used to convert every applicable futility bound separately. With the default cumulative type, an omitted target scale means conversion to the effect-estimate scale:

```
designPlan |>
    getFisherInformation() |>
    getFutilityBounds()
```

Alternatively, pipe `designPlan` directly into `getFutilityBounds()`; the latter then calls this function internally with the information type required by the requested target scale. Cumulative information can also be piped into a conversion to the design-specific treatment-effect scale:

```
designPlan |>
    getFisherInformation(type = "cumulative") |>
    getFutilityBounds(targetScale = "treatmentEffect")
```

In this case, the information object's type, stages, and originating design plan are validated before the endpoint-specific transformation is applied. For two-group rates, the Farrington–Manning inversion uses the design plan's group sizes and null-restricted rates directly. For Count Data, the design and stages are reused without recalculating Fisher information, while the exact negative-binomial inversion evaluates its variance at each candidate rate ratio from the design-specific sample-size and exposure assumptions.

Value

A `FisherInformation` R6 object. Its public fields are `information` (the calculated numeric value, vector, or matrix), `type`, `stage`, and `situations`. The latter contains descriptive labels such as the corresponding alternatives, event probabilities, hazard ratios, or count-data rates where available. The originating design plan is stored privately for subsequent pipe-based conversion with `getFutilityBounds()`.

Use `as.numeric()` to extract a plain numeric vector, `as.matrix()` to obtain a consistently arranged stage-by-situation matrix, and `as.data.frame()` to obtain a long-format data frame. Unsupported endpoint types are represented by `NA_real_` in the `information` field.

See Also

`getFutilityBounds()` for converting futility bounds using the calculated information.

Examples

```
## Not run:
designPlan <- getSampleSizeMeans(alternative = 0.4)
getFisherInformation(designPlan)

design <- getDesignGroupSequential(kMax = 3)
designPlan <- getPowerMeans(design,
    alternative = c(0.3, 0.4), maxNumberOfSubjects = 100
)
getFisherInformation(designPlan)
getFisherInformation(designPlan, stage = 2)
getFisherInformation(designPlan, stage = 1:3, type = "stageWise")

simulationResults <- getSimulationMeans(design,
    plannedSubjects = c(20, 40, 60), alternative = 0.4,
    maxNumberOfIterations = 10
)
getFisherInformation(simulationResults)

## End(Not run)
```

---

getFutilityBounds *Get Futility Bounds*

Description

This function converts futility bounds between different scales such as z-value, p-value, conditional power, predictive power, reverse conditional power, Wald effect estimate, and design-specific treatment effect.

Usage

```
getFutilityBounds(
  sourceValue,
  ...,
  sourceScale = c("zValue", "pValue", "conditionalPower", "condPowerAtObserved",
```

```
    "predictivePower", "reverseCondPower", "effectEstimate", "treatmentEffect"),
  targetScale = c("zValue", "pValue", "conditionalPower", "condPowerAtObserved",
    "predictivePower", "reverseCondPower", "effectEstimate", "treatmentEffect"),
  design = NULL,
  directionUpper = NA,
  theta = NA_real_,
  information = NA_real_,
  stage = NA_integer_,
  naAllowed = FALSE
)
```

Arguments

sourceValue	A numeric vector or matrix representing the futility bounds in the source scale. Alternatively, a trial design plan, simulation results, or a FisherInformation object returned by getFisherInformation() can be piped in; 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.
sourceScale	Character. The scale of the input futility bounds. Must be one of "zValue", "pValue", "conditionalPower", "condPowerAtObserved", "predictivePower", "reverseCondPower", "effectEstimate", or "treatmentEffect".
targetScale	Character. The scale to which the futility bounds should be converted. Must be one of "zValue", "pValue", "conditionalPower", "condPowerAtObserved", "predictivePower", "reverseCondPower", "effectEstimate", or "treatmentEffect".
design	The trial design. Required if either the sourceScale or targetScale is "reverseCondPower" or if the conversion involves conditional or predictive power in a group sequential or Fisher design; these conversions require a supported one-sided two-stage design. Conversion from or to "treatmentEffect" requires a trial design plan, rather than only its contained trial design, because endpoint-specific planning parameters are needed.
directionUpper	Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.
theta	Numeric. The assumed treatment effect under the alternative hypothesis on the unstandardized analysis scale. For example, in a survival design this is specified on the log hazard-ratio scale.
information	Numeric vector of length 1 or 2 specifying the information used in the conversion. In general, information[1] is the cumulative information available at the analysis to which the futility bound refers, whereas information[2] is the additional information planned after that analysis. The exact requirements depend on sourceScale and targetScale. A FisherInformation object can also be supplied; its type field must be consistent with the requested conversion; it must represent a single planning situation when used as this argument. See Details.
stage	Integer vector identifying the analysis stages represented by numeric sourceValue when converting from or to the "treatmentEffect" scale. It is inferred automatically from piped FutilityBounds or FisherInformation objects. If omitted otherwise, all interim stages are used.
naAllowed	Logical. Indicates if NA sourceValue are permitted. Default is FALSE.

Details

If the sourceScale and targetScale are the same, the function returns the input sourceValue without modification. Otherwise, the function is designed to convert between the specified scales.

Available scales

Each scale expresses the same interim futility threshold from a different perspective:

"zValue" The standardized interim test statistic. For directionUpper = TRUE, larger values favor the alternative; directionUpper = FALSE reverses this direction. Conversion between this scale and "pValue" does not require Fisher information.

"pValue" The one-sided p-value corresponding to the z-value and the selected direction. Smaller values indicate stronger evidence in favor of the alternative. Conversion between this scale and "zValue" does not require Fisher information.

"effectEstimate" The unstandardized, null-centered effect estimate $\widehat{\delta}$ on the Wald analysis scale, related to the z-value by $z = \widehat{\delta}\sqrt{I_1}$. Depending on the endpoint, this can be a mean or rate difference, or an effect on a transformed scale such as the log hazard-ratio scale. Adding the null value or applying a back-transformation may be necessary to obtain the endpoint's usual presentation scale. Fisher information is evaluated under the planning assumptions.

"treatmentEffect" The endpoint's natural treatment-effect scale, calculated with the same endpoint- and test-specific transformation used for futilityBoundsEffectScale in the design plan. For two-group binary rates this inverts the Farrington–Manning score statistic; for count data it uses the validated negative-binomial inversion with a candidate-dependent variance estimate; for survival data it returns the hazard-ratio scale; and other endpoints retain their existing design-specific transformations. This scale requires a trial design plan, either supplied as design or retained in a piped result. Cumulative Fisher information is calculated internally where the endpoint-specific transformation needs it, or taken from a piped FisherInformation object after validation. The Count Data inversion additionally uses the stage-specific sample size, exposure, recruitment, allocation, and overdispersion from the design plan.

"conditionalPower" The probability of rejecting the null hypothesis at the final analysis, conditional on the interim result and assuming the user-specified treatment effect theta for the future observations. It requires theta and the additional second-stage information.

"condPowerAtObserved" Conditional power calculated by using the interim effect estimate in place of a separately assumed effect. It treats the observed estimate as fixed when projecting the future data and, in the supported two-stage setting, depends on the information through the ratio of second-stage to first-stage information.

"predictivePower" The Bayesian predictive probability of rejecting the null hypothesis at the final analysis under a flat (improper) prior for the treatment effect. Unlike conditional power at the observed effect, it integrates uncertainty about that effect. In the supported two-stage setting, it also depends only on the information ratio.

"reverseCondPower" Reverse conditional power (also called reverse stochastic curtailment): the conditional probability that the interim result would be at least as unfavorable as the observed result, given that the final combined test statistic is at its critical boundary. It is independent of an assumed treatment effect and, for the supported inverse normal or group sequential setting, coincides with the predictive power based on a flat prior. Conversion between this scale and "zValue" or "pValue" uses the design's information rates, but does not require endpoint-specific Fisher information.

The four power-based scales take values between 0 and 1. Their values describe a futility threshold at an interim analysis; they should not be confused with the unconditional power of the trial design.

Piping design plans or Fisher information into getFutilityBounds

A trial design plan or simulation results object can be supplied directly. If the requested conversion needs Fisher information, `getFisherInformation()` is called internally with the required type: `"cumulative"` for an effect-estimate conversion and `"stageWise"` for conversions involving conditional or predictive power. It is not called for conversion from the design's z-value bounds to `"zValue"`, `"pValue"`, or `"reverseCondPower"`; the latter uses information rates from the design itself. This provides the short form

```
designPlan |>
    getFutilityBounds(targetScale = "condPowerAtObserved")
```

instead of explicitly inserting `getFisherInformation(type = "stageWise")` into the pipe.

A complete result of `getFisherInformation()` can be supplied as `sourceValue`, most conveniently with the base R pipe. In this form, `getFutilityBounds()` obtains the design, its z-value futility bounds, the requested stages, the information type, and the planning-situation labels from the fields of the `FisherInformation` object. If `targetScale` is omitted, the bounds are converted to `"effectEstimate"`; another target scale can be requested explicitly. If Fisher information is not needed for that scale, the object remains a valid source of the design and its futility bounds, but its information values are ignored and a warning is issued. The printed result then does not claim that Fisher information was used. An exception is `targetScale = "treatmentEffect"`: the cumulative information object is passed to and validated by the endpoint-specific transformation instead of being discarded. For Count Data, its originating design and requested stages are reused without recalculating Fisher information; the actual nonlinear inversion uses the corresponding design-specific sample sizes and a variance estimate evaluated at each candidate rate ratio.

Conversion to `"treatmentEffect"` uses `.getFutilityBoundsTreatmentEffectScale()` and therefore reproduces its validated endpoint-specific values exactly (up to the eight-digit rounding used for `futilityBoundsEffectScale`). This includes the nonlinear Count Data calculation and reproduces the corresponding design-plan field. Conversely, a result on this scale can be piped back into `getFutilityBounds()`; its retained design plan and stage metadata are used to recover the corresponding standardized bounds before conversion to the requested target scale. For two-group rates the output explicitly identifies the design-specific Farrington–Manning inversion. The distinct `"effectEstimate"` scale continues to report the null-centered Wald transformation based on cumulative Fisher information under the planning assumptions.

The conversion is performed separately for every planning situation and for every supplied interim analysis. Information for the final analysis is used where a conversion requires future information, but no final-stage futility bound is returned. Consequently, cumulative information is appropriate for the default conversion to the effect-estimate scale. Stage-wise information is required for conversions involving conditional or predictive power. An informative error is issued if its `type` field is incompatible with the requested target scale. Conversions to conditional power, conditional power at the observed effect, predictive power, or reverse conditional power remain restricted to one-sided two-stage designs. For a design with more than two stages, the conditional probability would additionally require a precise definition of how all remaining analyses and their stopping boundaries are handled. If a group sequential or inverse normal design contains only the default futility bound, that bound represents the absence of futility stopping. Its conversion to a conditional-power, predictive-power, or reverse-conditional- power scale is therefore returned as the exact boundary value `0`, without a numerical-range warning. This applies in both directions of the alternative.

Interpretation of information

The elements of `information` have different interpretations:

- `information[1]` is the cumulative Fisher information underlying the test statistic or effect estimate at the analysis where the futility bound is evaluated. In a two-stage design this is also the information contributed by the first stage.

- `information[2]` is the additional, non-cumulative Fisher information to be collected after that analysis. In a two-stage design this is the information contributed by the second stage, not the cumulative information at the second analysis.

Consequently, if I_1 and I_2 denote the cumulative information at the first and second analyses, respectively, specify `information = c(I1, I2 - I1)`. A single value is used for both elements.

Information required by conversion type

The required elements are determined by all scales involved in the conversion:

`"zValue"` **and** `"pValue"` Conversions between these two scales do not require information.

`"effectEstimate"` Requires `information[1]`. The standardized statistic and effect estimate are related by $z = \widehat{\theta}\sqrt{I_1}$. Thus the cumulative information at the analysis represented by the z-value or effect estimate must be supplied.

`"treatmentEffect"` Does not use the separately supplied numeric `information` argument. Instead, it requires a trial design plan and applies the endpoint- and test-specific transformation used by `futilityBoundsEffectScale`. Cumulative information is calculated internally when needed. Alternatively, a piped `FisherInformation` object of type `"cumulative"` is validated and used. For two-group rates, the validated Farrington–Manning calculation is based directly on the planned group sizes and null-restricted rates rather than on a generic Wald-information substitution. For Count Data, the validated negative-binomial inversion evaluates the variance at each candidate rate ratio and uses the planned stage sample size, recruitment, exposure, allocation ratio, and overdispersion. A piped cumulative `FisherInformation` object supplies and validates the design/stage context, but its stored numeric values do not replace that candidate-dependent variance calculation.

`"conditionalPower"` Requires `information[2]` together with `theta`. Here `information[2]` is the additional information available for the future stage over which conditional power is calculated.

`"condPowerAtObserved"` Requires both elements. The current effect is estimated using the cumulative information `information[1]` and projected over the additional future information `information[2]`.

`"predictivePower"` Requires both elements. Predictive power combines uncertainty based on the cumulative information already observed with the additional information planned for the future stage.

`"reverseCondPower"` Does not require an explicit `information` value; the required information fractions are taken from the specified `design`.

If `condPowerAtObserved` or `predictivePower` is involved and no complete information vector is supplied, the relative first- and second-stage information can be derived from an eligible two-stage `design` as `c(design$informationRates[1], 1 - design$informationRates[1])`. This normalization is sufficient for conversions that depend only on information ratios. An explicitly supplied vector is needed when absolute information is required, for example when conditional power is calculated under a specified value of `theta`.

A warning is issued if a two-element vector contains an information value not needed for the requested conversion. Set the unused element to `NA`, or pass a single value when using the same value for both elements is intended. A `FisherInformation` object returned by `getFisherInformation()` has a `type` field. `getFutilityBounds()` verifies that it is `"cumulative"` for an effect-estimate conversion or `"stageWise"` for a conditional- or predictive-power conversion, and stops with an error if the types do not match. Plain numeric input without type metadata remains supported for backward compatibility. If a `FisherInformation` object contains multiple planning situations, pipe it into `getFutilityBounds()` as `sourceValue`; the conversion is then performed separately for every situation. The `information` argument itself accepts such an object only when it represents one situation.

Obtaining information from a design plan

Use `getFisherInformation()` to calculate cumulative information from a design plan or simulation results object. For an analysis at stage `j`, its result can be passed directly as `information[1]` when converting between an effect estimate and a standardized statistic. To construct the information vector for a two-stage conditional- or predictive-power conversion, calculate the cumulative information at both analyses and use:

```
informationStage1 <- as.numeric(getFisherInformation(designPlan, stage = 1))
informationCumulative2 <- as.numeric(getFisherInformation(designPlan, stage = 2))
information <- c(
    informationStage1,
    informationCumulative2 - informationStage1
)
```

Value

A numeric vector or matrix of class `FutilityBounds` representing the futility bounds in the target scale, or `NULL` if the conversion is not implemented or yields no result. For pipe-based input with multiple stages and situations, rows represent interim stages and columns represent planning situations. The corresponding labels and calculation context are retained as attributes and displayed by `print.FutilityBounds()`.

See Also

`getFisherInformation()` for calculating the value of the `information` argument; `getDesignGroupSequential()`, `getDesignInverseNormal()`, `getDesignFisher()` for direct specification of futility bounds on different scales using the argument `futilityBoundsScale`.

Examples

```
## Not run:
# Example with identical source and target scales
getFutilityBounds(
    sourceValue = c(0, 0.5),
    sourceScale = "zValue",
    targetScale = "zValue"
)

# Example with different scales
getFutilityBounds(
   design = getDesignGroupSequential(kMax = 2, typeOfDesign = "noEarlyEfficacy", alpha = 0.05),
    information = c(10, 10),
    sourceValue = 0.5,
    sourceScale = "condPowerAtObserved",
    targetScale = "pValue"
)

# Pipe all planning situations into an effect-estimate-scale conversion
getDesignGroupSequential(
    informationRates = c(0.2, 0.7, 1),
    futilityBounds = c(-0.5, 0)
) |>
    getSampleSizeRates() |>
    getFisherInformation() |>
    getFutilityBounds()
```

```
# Fisher information is calculated internally with type = "stageWise"
getDesignGroupSequential(kMax = 2, futilityBounds = 0.3) |>
    getSampleSizeRates() |>
    getFutilityBounds(targetScale = "condPowerAtObserved")

# Reproduce the design-specific futility bounds on the treatment-effect scale
treatmentEffectBounds <- getDesignGroupSequential(
    informationRates = c(0.2, 0.7, 1),
    futilityBounds = c(0.3, 0.2)
) |>
    getSampleSizeRates() |>
    getFutilityBounds(targetScale = "treatmentEffect")

# Use treatmentEffect as a source scale; sourceScale is inferred here
treatmentEffectBounds |>
    getFutilityBounds(targetScale = "pValue")

## End(Not run)
```

`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 t_1, ..., t_kMax (that must be fixed prior to the trial), default is `(1:kMax) / kMax`. For the weighted inverse normal design, the weights are derived through w_1 = sqrt(t_1), and w_k = sqrt(t_k - t_(k-1)). For the weighted Fisher's combination test, the weights (scales) are w_k = sqrt((t_k - t_(k-1)) / t_1) (see the documentation).

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_2),..., 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_2),..., 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
That is, the output matrix of the function provides per stage (column) the cumulative probabilities for values specified in decisionMatrix and Inf, and reaching the stage, i.e., the test statistics is in the continuation region for the preceding stages. For 4 rows, the continuation region contains of two regions and the probability matrix is obtained analogously (cf., Wassmer and Brannath, 2025).

Value

Returns a numeric matrix containing the probabilities described in the details section.

See Also

Other design functions: getDesignCharacteristics(), getDesignConditionalDunnett(), getDesignFisher(), getDesignFixed(), getDesignGroupSequential(), getDesignInverseNormal(), getPowerAndAverageSampleNum

Examples

```
## Not run:
# Calculate Type I error rates in the two-sided group sequential setting when
# performing kMax 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, ])

## End(Not run)
```

`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 numeric 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

```
## Not run:
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)

## End(Not run)
```

`getObjects` *Get Objects from a Collection*

Description

S3 generic for retrieving the objects contained in a collection. The `TrialComparison` method delegates to `$getObjects()`.

The `TrialComparison` method returns the list of contained objects.

Usage

```
getObjects(x, ...)

## S3 method for class 'TrialComparison'
getObjects(x, ...)
```

Arguments

`x`	An object supporting a `getObjects()` method, such as a `TrialComparison`.
`...`	Further arguments. Currently unused for `TrialComparison`.

Details

The returned list contains references to the original R6 objects. Changing a contained object affects subsequent comparisons. Replacing or removing entries in the returned list does not change the comparison's membership; use `add()` and `delete()` to modify the collection.

Value

For `TrialComparison`, the list of contained objects in their stored order, retaining supplied names, or an empty list for an empty comparison.

See Also

`getComparison()`, `compare()`

Examples

```
comparison <- getComparison(
    fixed = getDesignFixed(),
    sequential = getDesignGroupSequential(kMax = 2)
)
objects <- comparison |> getObjects()
names(objects)
identical(objects$fixed, comparison$fixed)
```

`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 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 general estimates, information is calculated as the cumulative inverse squared standard error. 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.

Value

Returns a list that summarizes the observed information rates.

See Also

- `getAnalysisResults()` for using `getObservedInformationRates()` implicit,
- www.rpact.org/vignettes/planning/rpact_boundary_update_example

Examples

```
## Not run:
# 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
)

## End(Not run)
```

---

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

```
## Not run:
# 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")

## End(Not run)
```

`getPerformanceScore` *Get Performance Score*

Description

Calculates the conditional performance score, its sub-scores and components according to (Herrmann et al. (2020), doi:10.1002/sim.8534) and (Bokelmann et al. (2024), doi:10.1186/s12874024-021504) for a given simulation result from a two-stage design with continuous or binary endpoint. Larger (sub-)score and component values refer to a better performance.

Usage

```
getPerformanceScore(simulationResult)
```

Arguments

`simulationResult`
: A simulation result.

Details

The conditional performance score consists of two sub-scores, one for the sample size (subscoreSampleSize) and one for the conditional power (subscoreConditionalPower). Each of those are composed of a location (locationSampleSize, locationConditionalPower) and variation component (variationSampleSize, variationConditionalPower). The term conditional refers to an evaluation perspective where the interim results suggest a trial continuation with a second stage. The score can take values between 0 and 1. More details on the performance score can be found in Herrmann et al. (2020), doi:10.1002/sim.8534 and Bokelmann et al. (2024) doi:10.1186/s12874024021504.

Author(s)

Stephen Schueuerhuis

Examples

```
## Not run:
# Example from Table 3 in "A new conditional performance score for
# the evaluation of adaptive group sequential designs with samplesize
# recalculation from Herrmann et al 2023", p. 2097 for
# Observed Conditional Power approach and Delta = 0.5

# Create two-stage Pocock design with binding futility boundary at 0
design <- getDesignGroupSequential(
    kMax = 2, typeOfDesign = "P",
    futilityBounds = 0, bindingFutility = TRUE)

# Initialize sample sizes and effect;
# Sample sizes are referring to overall stage-wise sample sizes
n1 <- 100
n2 <- 100
nMax <- n1 + n2
alternative <- 0.5
```

```
# Perform Simulation; nMax * 1.5 defines the maximum
# sample size for the additional stage
simulationResult <- getSimulationMeans(
    design = design,
    normalApproximation = TRUE,
    thetaH0 = 0,
    alternative = alternative,
    plannedSubjects = c(n1, nMax),
    minNumberOfSubjectsPerStage = c(NA_real_, 1),
    maxNumberOfSubjectsPerStage = c(NA_real_, nMax * 1.5),
    conditionalPower = 0.8,
    directionUpper = TRUE,
    maxNumberOfIterations = 1e05,
    seed = 140
)

# Calculate performance score
getPerformanceScore(simulationResult)

## End(Not run)
```

`getPiecewiseHazardRatioArray` *Create an Array of Piecewise Hazard Ratios*

Description

Combines one or more group-by-interval hazard-ratio matrices into an array with dimensions group by interval by situation.

Usage

```
getPiecewiseHazardRatioArray(..., groupNames = NULL, intervalNames = NULL)
```

Arguments

`...`	Matrices, or one named list of matrices. Each matrix contains groups in rows and piecewise survival intervals in columns.
`groupNames`	Optional group names. Existing row names must agree in all matrices and with these names, including their order. Unnamed matrices are interpreted positionally.
`intervalNames`	Optional interval names.

Value

A numeric array with dimensions group by interval by situation.

`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). Must be a positive numeric value or vector: without `piecewiseSurvivalTime`, a vector defines alternative hazard rates to be considered; with `piecewiseSurvivalTime`, it contains one hazard rate per time interval. `getEventProbabilities()` uses only the first value of a non-piecewise vector.

`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). Must be a positive numeric of length 1, or a vector with one hazard rate per time interval if `piecewiseSurvivalTime` is specified.

`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. Must be a positive numeric value or vector; `getEventProbabilities()` accepts a single value only.

`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 `c(0.1, 0.2, 0.3)` (sample size calculations).

`pi2` A numeric value that represents the assumed event rate in the control group, default is `0.5`.

`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. Must be a positive numeric of length 1.

`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(q = time, shape = kappa, scale = 1 / lambda)` of the `stats` package.
For example, `getPiecewiseExponentialDistribution(time = 130, piecewiseLambda = 0.01, kappa = 4.2)` and `pweibull(q = 130, shape = 4.2, scale = 1 / 0.01)` provide the same 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

```
## Not run:
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, directionUpper = FALSE,
    maxNumberOfEvents = 40, maxNumberOfSubjects = 100)

# The object created by getPiecewiseSurvivalTime() can be used directly in
# getSimulationSurvival():
getSimulationSurvival(piecewiseSurvivalTime = pwst, directionUpper = FALSE,
    plannedEvents = 40, maxNumberOfSubjects = 100)

## End(Not run)
```

---

getPiecewiseSurvivalTimeEnrichment

Piecewise Exponential Enrichment Survival Time

Description

Creates a validated piecewise exponential survival specification for a patient-wise population-enrichment simulation.

Usage

```
getPiecewiseSurvivalTimeEnrichment(
  piecewiseSurvivalTime = NULL,
  lambdaControls = NULL,
  hazardRatio = NULL,
  medianControls = NULL,
  medianTreatments = NULL
)
```

Arguments

`piecewiseSurvivalTime`
: Numeric interval start times beginning with 0.

`lambdaControls`
: Non-negative, finite subset-by-interval control-hazard matrix. If both control hazards and hazard ratios have row names, their names and order must agree. Specify either `lambdaControls` or `medianControls`.

`hazardRatio`
: One subset-by-interval matrix, a named list of such matrices (one per situation), or a subset-by-interval-by-situation array. In the array, situation is the slowest-changing dimension. Specify either `hazardRatio` or `medianTreatments`.

`medianControls`
: Positive, finite exponential control medians in a subset-by-interval matrix, as an alternative to `lambdaControls`. Names and dimensions follow the same rules as for `lambdaControls`.

`medianTreatments`
: Positive, finite exponential treatment medians, in the same matrix, list, or array format as `hazardRatio`, as an alternative to it.

Details

At least two finite, strictly increasing interval starts are required. The last interval is open-ended. Hazards must be finite and non-negative; zero denotes an interval without events. Hazard ratios must be finite and strictly positive. Group and situation names must be unique and non-empty. Unnamed groups are positional; named groups must agree in order. Interval columns are positional and receive labels derived from the interval starts. Only piecewise exponential event times are supported; simulation requires `kappa = 1`. The returned time definition inherits from `ParameterSet` and supports `names()`, `print()`, `summary()`, `as.data.frame()`, `as.matrix()`, and `getObjectRCode()`. Hazard ratios specify the data-generating distribution, rather than separate interval-specific analysis estimands. Median inputs parameterize each interval's exponential hazard through `getLambdaByMedian(median, kappa = 1)`. They are not the actual medians of the complete piecewise survival distributions. The original interval medians are retained as user-defined parameters; hazards and hazard ratios calculated from them are marked as derived. No automatic recycling across intervals or groups is performed.

Value

A `PiecewiseSurvivalTimeEnrichment` object.

Examples

```
piecewiseTime <- getPiecewiseSurvivalTimeEnrichment(
    piecewiseSurvivalTime = c(0, 6),
    lambdaControls = rbind(
        markerPositive = c(0.08, 0.12),
```

```
        markerNegative = c(0.05, 0.09)
    ),
    hazardRatio = list(
        favorable = rbind(
            markerPositive = c(0.65, 0.55),
            markerNegative = c(0.9, 0.8)
        ),
        conservative = rbind(
            markerPositive = c(0.8, 0.7),
            markerNegative = c(1, 0.9)
        )
    )
)
as.data.frame(piecewiseTime)

getPiecewiseSurvivalTimeEnrichment(
    piecewiseSurvivalTime = c(0, 6),
    medianControls = rbind(markerPositive = c(12, 24), markerNegative = c(18, 30)),
    medianTreatments = rbind(markerPositive = c(18, 36), markerNegative = c(20, 32))
)
```

`getPiecewiseSurvivalTimeMultiArm`
Piecewise Exponential Multi-Arm Survival Time

Description

Creates a validated piecewise exponential survival specification for a patient-wise multi-arm simulation.

Usage

```
getPiecewiseSurvivalTimeMultiArm(
  piecewiseSurvivalTime = NULL,
  lambdaControl = NULL,
  hazardRatio = NULL,
  medianControl = NULL,
  medianTreatments = NULL
)
```

Arguments

`piecewiseSurvivalTime`	Numeric interval start times beginning with 0.
`lambdaControl`	Non-negative, finite control hazards, one per interval. Specify either `lambdaControl` or `medianControl`.
`hazardRatio`	One active-arm-by-interval matrix, a named list of such matrices (one per situation), or an active-arm-by-interval-by-situation array. In the array, situation is the slowest-changing dimension. Specify either `hazardRatio` or `medianTreatments`.
`medianControl`	Positive, finite exponential control medians, one per interval, as an alternative to `lambdaControl`. Values are not recycled.

medianTreatments
: Positive, finite exponential treatment medians, in the same matrix, list, or array format as hazardRatio, as an alternative to it.

Details

At least two finite, strictly increasing interval starts are required. The last interval is open-ended. Hazards must be finite and non-negative; zero denotes an interval without events. Hazard ratios must be finite and strictly positive. Group and situation names must be unique and non-empty. Unnamed groups are positional; named groups must agree in order. Interval columns are positional and receive labels derived from the interval starts. Only piecewise exponential event times are supported; simulation requires kappa = 1. The returned time definition inherits from ParameterSet and supports names(), print(), summary(), as.data.frame(), as.matrix(), and getObjectRCode(). Hazard ratios specify the data-generating distribution, rather than separate interval-specific analysis estimands. Median inputs parameterize each interval's exponential hazard through getLambdaByMedian(median, kappa = 1). They are not the actual medians of the complete piecewise survival distributions. The original interval medians are retained as user-defined parameters; hazards and hazard ratios calculated from them are marked as derived. No automatic recycling across intervals or groups is performed.

Value

A PiecewiseSurvivalTimeMultiArm object.

Examples

```
hazardRatio <- getPiecewiseHazardRatioArray(
    favorable = rbind(lowDose = c(0.9, 0.8), highDose = c(0.8, 0.6)),
    conservative = rbind(lowDose = c(1, 0.9), highDose = c(0.9, 0.75))
)
piecewiseTime <- getPiecewiseSurvivalTimeMultiArm(
    piecewiseSurvivalTime = c(0, 6),
    lambdaControl = c(0.08, 0.12),
    hazardRatio = hazardRatio
)
piecewiseTime

getPiecewiseSurvivalTimeMultiArm(
    piecewiseSurvivalTime = c(0, 6),
    medianControl = c(12, 24),
    medianTreatments = rbind(lowDose = c(15, 30), highDose = c(18, 36))
)
```

---

getPowerAndAverageSampleNumber

Get Power And Average Sample Number

Description

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

Usage

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

Arguments

design	The trial design.
...	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
theta	A vector of standardized effect sizes (theta values), default is a sequence from -1 to 1.
nMax	The maximum sample size. Must be a positive integer of length 1.

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()`, `getDesignFixed()`, `getDesignGroupSequential()`, `getDesignInverseNormal()`, `getGroupSequentialProbabili`

Examples

```
## Not run:
# 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)

## End(Not run)
```

`getPowerCounts` *Get Power Counts*

Description

Returns the power, stopping probabilities, and expected sample size for testing mean rates for negative binomial distributed event numbers in two samples at given sample sizes.

Usage

```
getPowerCounts(
  design = NULL,
  ...,
  directionUpper = NA,
  maxNumberOfSubjects = NA_real_,
  lambda1 = NA_real_,
  lambda2 = NA_real_,
  lambda = NA_real_,
  theta = NA_real_,
  thetaH0 = 1,
  overdispersion = 0,
  fixedExposureTime = NA_real_,
  accrualTime = NA_real_,
  accrualIntensity = NA_real_,
  followUpTime = 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.
`directionUpper`	Logical. Specifies the direction of the alternative, only applicable for one-sided testing; `TRUE` means that larger values of the test statistics yield smaller p-values. The default is `TRUE` for means, rates, and count data, and `FALSE` for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified `directionUpper` must agree with it.

maxNumberOfSubjects

maxNumberOfSubjects > 0 needs to be specified for power calculations or calculation of necessary follow-up (count data). For two treatment arms, it is the maximum number of subjects for both treatment arms.

lambda1 A numeric value or vector that represents the assumed rate of a homogeneous Poisson process in the active treatment group, there is no default.

lambda2 A numeric value that represents the assumed rate of a homogeneous Poisson process in the control group, there is no default.

lambda A numeric value or vector that represents the assumed rate of a homogeneous Poisson process in the pooled treatment groups, there is no default.

theta A numeric value or vector that represents the assumed mean ratios lambda1/lambda2 of a homogeneous Poisson process, there is no default.

thetaH0 The null hypothesis value, default is 0 for general estimates, the normal case, and the binary case, 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

- *general estimates*: a value on the scale of the supplied estimates can be specified.
- *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.
- *count data*: a bound for testing H0: lambda1 / lambda2 = 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.

overdispersion A numeric value that represents the assumed overdispersion of the negative binomial distribution, default is 0. Must be >= 0.

fixedExposureTime

If specified, the fixed time of exposure per subject for count data, there is no default. For power calculations and simulations, exactly one of fixedExposureTime and followUpTime must be specified.

accrualTime If specified, the assumed accrual time interval(s) for the study, there is no default. A single value defines the end of accrual; if a vector is specified, its first value must be 0. If followUpTime is specified, accrualTime can have at most two elements. Required for getSimulationCounts().

accrualIntensity

If specified, the assumed accrual intensities for the study, there is no default.

followUpTime If specified, the assumed (additional) follow-up time for the study, there is no default. The total study duration is accrualTime + followUpTime. For power calculations and simulations, exactly one of fixedExposureTime and followUpTime must be specified.

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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.

Details

At given design the function calculates the power, stopping probabilities, and expected sample size for testing the ratio of two mean rates of negative binomial distributed event numbers in two samples at given maximum sample size and effect. The power calculation is performed either for a fixed exposure time or a variable exposure time with fixed follow-up where the information over the stages is calculated according to the specified information rate in the design. Additionally, an allocation ratio = n1 / n2 can be specified where n1 and n2 are the number of subjects in the two treatment groups. A null hypothesis value thetaH0 can also be specified.

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(), getPowerRates(), getPowerSurvival()

Examples

```
## Not run:
# Fixed sample size trial where a therapy is assumed to decrease the
# exacerbation rate from 1.4 to 1.05 (25% decrease) within an
# observation period of 1 year, i.e., each subject has a equal
# follow-up of 1 year.
# Calculate power at significance level 0.025 at given sample size = 180
# for a range of lambda1 values if the overdispersion is assumed to be
# equal to 0.5, is obtained by
getPowerCounts(alpha = 0.025, lambda1 = seq(1, 1.4, 0.05), lambda2 = 1.4,
    maxNumberOfSubjects = 180, overdispersion = 0.5, fixedExposureTime = 1)
```

```
# Group sequential alpha and beta spending function design with O'Brien and
# Fleming type boundaries: Power and test characteristics for N = 286,
# under the assumption of a fixed exposure time, and for a range of
# lambda1 values:
getPowerCounts(design = getDesignGroupSequential(
        kMax = 3, alpha = 0.025, beta = 0.2,
        typeOfDesign = "asOF", typeBetaSpending = "bsOF"),
    lambda1 = seq(0.17, 0.23, 0.01), lambda2 = 0.3,
    directionUpper = FALSE, overdispersion = 1, maxNumberOfSubjects = 286,
    fixedExposureTime = 12, accrualTime = 6)

# Group sequential design alpha spending function design with O'Brien and
# Fleming type boundaries: Power and test characteristics for N = 1976,
# under variable exposure time with uniform recruitment over 1.25 months,
# study time (accrual + followup) = 4 (lambda1, lambda2, and overdispersion
# as specified, no futility stopping):
getPowerCounts(design = getDesignGroupSequential(
        kMax = 3, alpha = 0.025, beta = 0.2, typeOfDesign = "asOF"),
    lambda1 = seq(0.08, 0.09, 0.0025), lambda2 = 0.125,
    overdispersion = 5, directionUpper = FALSE, maxNumberOfSubjects = 1976,
    followUpTime = 2.75, accrualTime = 1.25)

## End(Not run)
```

`getPowerMeans` *Get Power Means*

Description

Returns the power, stopping probabilities, and expected sample size for testing means in one or two samples at given maximum 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 power for one-sided testing of H0: mu1 / mu2 = thetaH0 is calculated, default is FALSE.
thetaH0	The null hypothesis value, default is 0 for general estimates, the normal case, and the binary case, 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 • *general estimates*: a value on the scale of the supplied estimates can be specified. • *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. • *count data*: a bound for testing H0: lambda1 / lambda2 = 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. For two-armed trials, it is allowed to specify the standard deviations separately, i.e., as vector with two elements. If meanRatio = TRUE is specified, stDev defines the coefficient of variation sigma / mu2.
directionUpper	Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.
maxNumberOfSubjects	maxNumberOfSubjects > 0 needs to be specified for power calculations or calculation of necessary follow-up (count data). 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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.

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 where n1 and n2 are the number of subjects in the two treatment groups. 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: getPowerCounts(), getPowerRates(), getPowerSurvival()

Examples

```
## Not run:
# 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)

## End(Not run)
```

`getPowerRates` *Get Power Rates*

Description

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

Usage

```
getPowerRates(
  design = NULL,
  ...,
  groups = 2L,
  riskRatio = FALSE,
  thetaH0 = ifelse(riskRatio, 1, 0),
  pi1 = NA_real_,
  pi2 = NA_real_,
  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 general estimates, the normal case, and the binary case, 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 • *general estimates*: a value on the scale of the supplied estimates can be specified.

- *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.
- *count data*: a bound for testing H0: `lambda1 / lambda2 = 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` Logical. Specifies the direction of the alternative, only applicable for one-sided testing; `TRUE` means that larger values of the test statistics yield smaller p-values. The default is `TRUE` for means, rates, and count data, and `FALSE` for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified `directionUpper` must agree with it.

`maxNumberOfSubjects`

`maxNumberOfSubjects > 0` needs to be specified for power calculations or calculation of necessary follow-up (count data). 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. For simulating means and rates for a two treatment groups design, it can be a vector of length `kMax`, the number of stages. It can be a vector of length `kMax`, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally `allocationRatioPlanned` is treated as a vector of length `kMax`, not a scalar.

Details

At given design the function calculates the power, stopping probabilities, and expected sample size for testing rates at 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 where `n1` and `n2` are the number of subjects in the two treatment groups. 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: getPowerCounts(), getPowerMeans(), getPowerSurvival()

Examples

```
## Not run:
# 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)

## End(Not run)
```

`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.

Planning conditions on deterministic piecewise constant accrual and planned allocation, retaining fractional accrued counts without rounding. Within each accrual interval the individual competing-risk dropout incidence is averaged over enrollment dates and weighted by planned enrollment by the horizon. The denominator includes only subjects accrued by that time. The horizon is the corresponding stage's planned analysis, without early-stopping weighting. An unavailable horizon (including calculations without observation times) or no accrued subjects gives NA; otherwise zero dropout gives zero. Exponential and piecewise exponential event times are supported. The existing restriction against Weibull event times with nonzero dropout remains in place.

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 general estimates, the normal case, and the binary case, 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

- *general estimates*: a value on the scale of the supplied estimates can be specified.
- *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.
- *count data*: a bound for testing H0: `lambda1 / lambda2 = 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` Logical. Specifies the direction of the alternative, only applicable for one-sided testing; `TRUE` means that larger values of the test statistics yield smaller p-values. The default is `TRUE` for means, rates, and count data, and `FALSE` for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified `directionUpper` must agree with it.

`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 `c(0.1, 0.2, 0.3)` (sample size calculations).

`pi2` A numeric value that represents the assumed event rate in the control group, default is `0.5`.

`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). Must be a positive numeric value or vector: without `piecewiseSurvivalTime`, a vector defines alternative hazard rates to be considered; with `piecewiseSurvivalTime`, it contains one hazard rate per time interval. `getEventProbabilities()` uses only the first value of a non-piecewise vector.

`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). Must be a positive numeric of length 1, or a vector with one hazard rate per time interval if `piecewiseSurvivalTime` is specified.

`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. Must be a positive numeric of length 1.

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(q = time, shape = kappa, scale = 1 / lambda) of the stats package.
For example, getPiecewiseExponentialDistribution(time = 130, piecewiseLambda = 0.01, kappa = 4.2) and pweibull(q = 130, shape = 4.2, scale = 1 / 0.01) provide the same 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. Must be a positive numeric value or vector; getEventProbabilities() accepts a single value only.

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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.

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 numeric 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 are specified, this will be calculated. Must be a positive integer of length 1.

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. For multi-arm simulations, this can be a vector of length equal to the number of active arms, specifying different drop-out rates for each active arm.

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.

Accrual averages use analytic integration, split at event-hazard boundaries. When both input dropout rates are zero, the proportions are retained in the result object but omitted from print and summary output.

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: getPowerCounts(), getPowerMeans(), getPowerRates()

Examples

```
## Not run:
# Fixed sample size with minimum required definitions, pi1 = c(0.2, 0.3, 0.4, 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(0.2, 0.3, 0.4, 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)

## End(Not run)
```

`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 survival `SimulationResults` object created by `getSimulationSurvival()`, `getSimulationMultiArmSurvival()`, or `getSimulationEnrichmentSurvival()`.
`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 the simulation function was called with `maxNumberOfRawDatasetsPerStage` > 0 (default is `0`). Multi-arm and enrichment simulations must use a patient-wise simulation type.

This function can be used to get the simulated raw data from a simulation results object obtained from an ordinary, multi-arm, or enrichment survival simulation. 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. `timeUnderObservation`: The time under observation is defined as follows:

```
if (event == TRUE) {
    timeUnderObservation <- survivalTime
} else if (dropoutEvent == TRUE) {
    timeUnderObservation <- dropoutTime
} else {
    timeUnderObservation <- analysisTime - accrualTime
}
```

where `analysisTime` is available from the corresponding row of `getData()`.

9. `event`: `TRUE` if an event occurred; `FALSE` otherwise.
10. `dropoutEvent`: `TRUE` if an dropout event occurred; `FALSE` otherwise.

Multi-arm and enrichment raw data additionally contain a stable integer `scenario` identifier. Enrichment raw data also contain the subject's `subGroup`. For these simulation types, `aggregate = TRUE` aggregates multi-arm data by scenario and iteration and enrichment data by scenario, iteration, and subgroup.

Value

Returns a `data.frame`.

Examples

```
## Not run:
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)

## End(Not run)
```

---

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 = NA,
  tolerance = 1e-06,
  stage = NA_integer_
)
```

Arguments

design	The trial design.
dataInput	The summary data used for calculating the test results. This is an element of DatasetGeneral, DatasetMeans, DatasetRates, or 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	Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.
tolerance	The numerical tolerance, default is 1e-06. Must be a positive numeric of length 1.
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(), getClosedConditionalD
getConditionalPower(), getConditionalRejectionProbabilities(), getFinalConfidenceInterval(),
getFinalPValue(), getRepeatedPValues(), getStageResults(), getTestActions()

Examples

```
## Not run:
design <- getDesignInverseNormal(kMax = 2)
data <- getDataset(
    n      = c( 20,  30),
    means  = c( 50,  51),
    stDevs = c(130, 140)
)
getRepeatedConfidenceIntervals(design, dataInput = data)

## End(Not run)
```

---

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. Must be a positive numeric of length 1.

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

Repeated p-values are calculated by identifying the smallest global significance level for which the observed test statistic would lead to rejection under the same design rule. Therefore, they require that the design can be re-created for arbitrary values of the global significance level. For user-defined designs with fixed userAlphaSpending values, this alpha-dependent design family is not uniquely defined. Consequently, repeated p-values are not available for such designs. Repeated p-values are also unavailable for sequential designs with typeBetaSpending = "bsUser", regardless of whether futility is binding. These calculations return NA with a warning; other analysis results remain available.

See Also

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

Examples

```
## Not run:
design <- getDesignInverseNormal(kMax = 2)
data <- getDataset(
    n      = c( 20,  30),
    means  = c( 50,  51),
    stDevs = c(130, 140)
)
getRepeatedPValues(getStageResults(design, dataInput = data))

## End(Not run)
```

---

getSampleSizeCounts *Get Sample Size Counts*

Description

Returns the sample size for testing the ratio of mean rates of negative binomial distributed event numbers in two samples at given effect.

Usage

```
getSampleSizeCounts(
  design = NULL,
  ...,
  directionUpper = NA,
  lambda1 = NA_real_,
  lambda2 = NA_real_,
  lambda = NA_real_,
```

```
  theta = NA_real_,
  thetaH0 = 1,
  overdispersion = 0,
  fixedExposureTime = NA_real_,
  accrualTime = NA_real_,
  accrualIntensity = NA_real_,
  followUpTime = NA_real_,
  maxNumberOfSubjects = NA_integer_,
  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.
directionUpper	Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.
lambda1	A numeric value or vector that represents the assumed rate of a homogeneous Poisson process in the active treatment group, there is no default.
lambda2	A numeric value that represents the assumed rate of a homogeneous Poisson process in the control group, there is no default.
lambda	A numeric value or vector that represents the assumed rate of a homogeneous Poisson process in the pooled treatment groups, there is no default.
theta	A numeric value or vector that represents the assumed mean ratios lambda1/lambda2 of a homogeneous Poisson process, there is no default.
thetaH0	The null hypothesis value, default is 0 for general estimates, the normal case, and the binary case, 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 • *general estimates*: a value on the scale of the supplied estimates can be specified. • *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. • *count data*: a bound for testing H0: lambda1 / lambda2 = 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.

`overdispersion` A numeric value that represents the assumed overdispersion of the negative binomial distribution, default is `0`. Must be `>= 0`.

`fixedExposureTime`
If specified, the fixed time of exposure per subject for count data, there is no default. For power calculations and simulations, exactly one of `fixedExposureTime` and `followUpTime` must be specified.

`accrualTime` If specified, the assumed accrual time interval(s) for the study, there is no default. A single value defines the end of accrual; if a vector is specified, its first value must be `0`. If `followUpTime` is specified, `accrualTime` can have at most two elements. Required for `getSimulationCounts()`.

`accrualIntensity`
If specified, the assumed accrual intensities for the study, there is no default.

`followUpTime` If specified, the assumed (additional) follow-up time for the study, there is no default. The total study duration is `accrualTime + followUpTime`. For power calculations and simulations, exactly one of `fixedExposureTime` and `followUpTime` must be specified.

`maxNumberOfSubjects`
`maxNumberOfSubjects > 0` needs to be specified for power calculations or calculation of necessary follow-up (count data). 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`. 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 information, and stage-wise and maximum sample size for testing mean rates of negative binomial distributed event numbers in two samples at given effect. The sample size calculation is performed either for a fixed exposure time or a variable exposure time with fixed follow-up. For the variable exposure time case, at given maximum sample size the necessary follow-up time is calculated. The planned calendar time of interim stages is calculated if an accrual time is defined. Additionally, an allocation ratio = `n1 / n2` can be specified where `n1` and `n2` are the number of subjects in the two treatment groups. A null hypothesis value `thetaH0` can also be specified.

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(), getSampleSizeRates(), getSampleSizeSurvival()

Examples

```
## Not run:
# Fixed sample size trial where a therapy is assumed to decrease the
# exacerbation rate from 1.4 to 1.05 (25% decrease) within an observation
# period of 1 year, i.e., each subject has an equal follow-up of 1 year.
# The sample size that yields 90% power at significance level 0.025 for
# detecting such a difference, if the overdispersion is assumed to be
# equal to 0.5, is obtained by
getSampleSizeCounts(alpha = 0.025, beta = 0.1, lambda2 = 1.4,
    theta = 0.75, overdispersion = 0.5, fixedExposureTime = 1)

# Noninferiority test with blinded sample size reassessment to reproduce
# Table 2 from Friede and Schmidli (2010):
getSampleSizeCounts(alpha = 0.025, beta = 0.2, lambda = 1, theta = 1,
    thetaH0 = 1.15, overdispersion = 0.4, fixedExposureTime = 1)

# Group sequential alpha and beta spending function design with O'Brien and
# Fleming type boundaries: Estimate observation time under uniform
# recruitment of patients over 6 months and a fixed exposure time of 12
# months (lambda1, lambda2, and overdispersion as specified):
getSampleSizeCounts(design = getDesignGroupSequential(
        kMax = 3, alpha = 0.025, beta = 0.2,
        typeOfDesign = "asOF", typeBetaSpending = "bsOF"),
    lambda1 = 0.2, lambda2 = 0.3, overdispersion = 1,
    fixedExposureTime = 12, accrualTime = 6)

# Group sequential alpha spending function design with O'Brien and Fleming
# type boundaries: Sample size for variable exposure time with uniform
# recruitment over 1.25 months and study time (accrual + followup) = 4
# (lambda1, lambda2, and overdispersion as specified, no futility stopping):
getSampleSizeCounts(design = getDesignGroupSequential(
        kMax = 3, alpha = 0.025, beta = 0.2, typeOfDesign = "asOF"),
    lambda1 = 0.0875, lambda2 = 0.125, overdispersion = 5,
    followUpTime = 2.75, accrualTime = 1.25)

## End(Not run)
```

---

getSampleSizeMeans *Get Sample Size Means*

Description

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

Usage

```
getSampleSizeMeans(
  design = NULL,
  ...,
  groups = 2L,
  normalApproximation = FALSE,
  meanRatio = FALSE,
  thetaH0 = ifelse(meanRatio, 1, 0),
  alternative = seq(0.2, 1, 0.2),
  stDev = 1,
  directionUpper = NA,
  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 general estimates, the normal case, and the binary case, 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 • *general estimates*: a value on the scale of the supplied estimates can be specified. • *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. • *count data*: a bound for testing H0: `lambda1 / lambda2 = 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`. For two-armed trials, it is allowed to specify the standard deviations separately, i.e., as vector with two elements. If `meanRatio = TRUE` is specified, `stDev` defines the coefficient of variation `sigma / mu2`.

`directionUpper` Logical. Specifies the direction of the alternative, only applicable for one-sided testing; `TRUE` means that larger values of the test statistics yield smaller p-values. The default is `TRUE` for means, rates, and count data, and `FALSE` for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified `directionUpper` must agree with it.

`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 and maximum sample size for testing means. In a two treatment groups design, additionally, an allocation ratio = `n1 / n2` can be specified where `n1` and `n2` are the number of subjects in the two treatment groups. 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: `getSampleSizeCounts()`, `getSampleSizeRates()`, `getSampleSizeSurvival()`

Examples

```
## Not run:
# 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)

# Calculate sample sizes in a three-stage Pocock two-armed design testing
# H0: mu = 2 for a range of alternative values 3,4,5 with assumed standard
# deviations = 3 and 4, respectively, in the two groups of observations;
# one-sided alpha = 0.05, power 1 - beta = 90%:
getSampleSizeMeans(getDesignGroupSequential(typeOfDesign = "P", alpha = 0.05,
    sided = 1, beta = 0.1), groups = 2,
    alternative = seq(3, 5, 1), stDev = c(3, 4))

## End(Not run)
```

---

getSampleSizeRates *Get Sample Size Rates*

Description

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

Usage

```
getSampleSizeRates(
  design = NULL,
  ...,
  groups = 2L,
  normalApproximation = TRUE,
  conservative = TRUE,
  riskRatio = FALSE,
  thetaH0 = ifelse(riskRatio, 1, 0),
  pi1 = NA_real_,
  pi2 = NA_real_,
  directionUpper = NA,
  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.
conservative	For the case of one treatment group and normalApproximation = FALSE, if TRUE, the sample size is calculated such that for larger sample size than the calculated, the power is larger than 1 - beta, for conservative = FALSE, the minimum sample size, for which power exceeds 1 - beta is calculated, 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 general estimates, the normal case, and the binary case, 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 • *general estimates*: a value on the scale of the supplied estimates can be specified. • *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. • *count data*: a bound for testing H0: lambda1 / lambda2 = 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	Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.

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 and maximum sample size for testing rates. In a two treatment groups design, additionally, an allocation ratio = n1 / n2 can be specified where n1 and n2 are the number of subjects in the two treatment groups. 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 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: getSampleSizeCounts(), getSampleSizeMeans(), getSampleSizeSurvival()

Examples

```
## Not run:
# 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)

## End(Not run)
```

`getSampleSizeSurvival` *Get Sample Size Survival*

Description

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

Planning conditions on deterministic piecewise constant accrual and planned allocation, retaining fractional accrued counts without rounding. Within each accrual interval the individual competing-risk dropout incidence is averaged over enrollment dates and weighted by planned enrollment by the horizon. The denominator includes only subjects accrued by that time. The horizon is the corresponding stage's planned analysis, without early-stopping weighting. An unavailable horizon (including calculations without observation times) or no accrued subjects gives NA; otherwise zero dropout gives zero. Exponential and piecewise exponential event times are supported. The existing restriction against Weibull event times with nonzero dropout remains in place.

Usage

```
getSampleSizeSurvival(
  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 = 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 general estimates, the normal case, and the binary case, 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

- *general estimates*: a value on the scale of the supplied estimates can be specified.
- *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.
- *count data*: a bound for testing H0: `lambda1 / lambda2 = 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` Logical. Specifies the direction of the alternative, only applicable for one-sided testing; `TRUE` means that larger values of the test statistics yield smaller p-values. The default is `TRUE` for means, rates, and count data, and `FALSE` for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified `directionUpper` must agree with it.

`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 `c(0.1, 0.2, 0.3)` (sample size calculations).

`pi2` A numeric value that represents the assumed event rate in the control group, default is `0.5`.

`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). Must be a positive numeric value or vector: without `piecewiseSurvivalTime`, a vector defines alternative hazard rates to be considered; with `piecewiseSurvivalTime`, it contains one hazard rate per time interval. `getEventProbabilities()` uses only the first value of a non-piecewise vector.

`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). Must be a positive numeric of length 1, or a vector with one hazard rate per time interval if `piecewiseSurvivalTime` is specified.

`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. Must be a positive numeric of length 1.

`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(q = time, shape = kappa, scale = 1 / lambda)` of the `stats` package.
For example, `getPiecewiseExponentialDistribution(time = 130, piecewiseLambda = 0.01, kappa = 4.2)` and `pweibull(q = 130, shape = 4.2, scale = 1 / 0.01)` provide the same 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. Must be a positive numeric value or vector; `getEventProbabilities()` accepts a single value only.

`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 numeric 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`. For multi-arm simulations, this can be a vector of length equal to the number of active arms, specifying different drop-out rates for each active arm.

`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.

Accrual averages use analytic integration, split at event-hazard boundaries. When both input dropout rates are zero, the proportions are retained in the result object but omitted from print and summary output.

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: getSampleSizeCounts(), getSampleSizeMeans(), getSampleSizeRates()

Examples

```
## Not run:
# 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)

## End(Not run)
```

`getSimulationCounts` *Get Simulation Counts*

Description

Returns the simulated power, stopping probabilities, conditional power, and expected sample size for testing mean rates for negative binomial distributed event numbers in the two treatment groups testing situation.

Usage

```
getSimulationCounts(
  design = NULL,
  ...,
  plannedCalendarTime = NA_real_,
  maxNumberOfSubjects = NA_real_,
  lambda1 = NA_real_,
  lambda2 = NA_real_,
  lambda = NA_real_,
  theta = NA_real_,
  directionUpper = NA,
  thetaH0 = 1,
  overdispersion = 0,
  fixedExposureTime = NA_real_,
  accrualTime = NA_real_,
  accrualIntensity = NA_real_,
  followUpTime = NA_real_,
  allocationRatioPlanned = NA_real_,
  maxNumberOfIterations = NA_integer_,
  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.

`plannedCalendarTime`
For simulating count data, the time points where an analysis is planned to be performed. Required for group sequential designs as a vector of length `kMax`; ignored for a fixed design.

`maxNumberOfSubjects`
`maxNumberOfSubjects > 0` needs to be specified for power calculations or calculation of necessary follow-up (count data). For two treatment arms, it is the maximum number of subjects for both treatment arms.

`lambda1` A numeric value or vector that represents the assumed rate of a homogeneous Poisson process in the active treatment group, there is no default.

`lambda2` A numeric value that represents the assumed rate of a homogeneous Poisson process in the control group, there is no default.

`lambda` A numeric value or vector that represents the assumed rate of a homogeneous Poisson process in the pooled treatment groups, there is no default.

`theta` A numeric value or vector that represents the assumed mean ratios lambda1/lambda2 of a homogeneous Poisson process, there is no default.

`directionUpper` Logical. Specifies the direction of the alternative, only applicable for one-sided testing; `TRUE` means that larger values of the test statistics yield smaller p-values. The default is `TRUE` for means, rates, and count data, and `FALSE` for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified `directionUpper` must agree with it.

`thetaH0` The null hypothesis value, default is `0` for general estimates, the normal case, and the binary case, 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

- *general estimates*: a value on the scale of the supplied estimates can be specified.
- *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.
- *count data*: a bound for testing H0: `lambda1 / lambda2 = 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`.

`overdispersion` A numeric value that represents the assumed overdispersion of the negative binomial distribution, default is `0`. Must be `>= 0`.

fixedExposureTime
: If specified, the fixed time of exposure per subject for count data, there is no default. For power calculations and simulations, exactly one of `fixedExposureTime` and `followUpTime` must be specified.

accrualTime
: If specified, the assumed accrual time interval(s) for the study, there is no default. A single value defines the end of accrual; if a vector is specified, its first value must be `0`. If `followUpTime` is specified, `accrualTime` can have at most two elements. Required for `getSimulationCounts()`.

accrualIntensity
: If specified, the assumed accrual intensities for the study, there is no default.

followUpTime
: If specified, the assumed (additional) follow-up time for the study, there is no default. The total study duration is `accrualTime` + `followUpTime`. For power calculations and simulations, exactly one of `fixedExposureTime` and `followUpTime` must be specified.

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. For simulating means and rates for a two treatment groups design, it can be a vector of length `kMax`, the number of stages. It can be a vector of length `kMax`, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally `allocationRatioPlanned` is treated as a vector of length `kMax`, not a scalar.

maxNumberOfIterations
: The number of simulation iterations, default is `1000`. Must be a positive integer of length 1.

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

showStatistics
: Logical. 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` and a null hypothesis value `thetaH0` can be specified.

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.

`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. `lambda1`: The assumed or derived event rate in the treatment group.
4. `lambda2`: The assumed or derived event rate in the control group.
5. `accrualTime`: The assumed accrualTime.
6. `followUpTime`: The assumed followUpTime.
7. `overdispersion`: The assumed overdispersion.
8. `fixedFollowUp`: The assumed fixedFollowUp.
9. `numberOfSubjects`: The number of subjects under consideration when the (interim) analysis takes place.
10. `rejectPerStage`: 1 if null hypothesis can be rejected, 0 otherwise.
11. `futilityPerStage`: 1 if study should be stopped for futility, 0 otherwise.
12. `testStatistic`: The test statistic that is used for the test decision
13. `estimatedLambda1`: The estimated rate in treatment group 1.
14. `estimatedLambda2`: The estimated rate in treatment group 2.
15. `estimatedOverdispersion`: The estimated overdispersion.
16. `infoAnalysis`: The Fisher information at interim stage.
17. `trialStop`: `TRUE` if study should be stopped for efficacy or futility or final stage, `FALSE` otherwise.
18. `conditionalPowerAchieved`: Not yet available

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

```
## Not run:
# Fixed sample size design with two groups, fixed exposure time
getSimulationCounts(
    theta = 1.8,
    lambda2 = 0.2,
    maxNumberOfSubjects = 200,
```

```
    plannedCalendarTime = 8,
    maxNumberOfIterations = 1000,
    fixedExposureTime = 6,
    accrualTime = 3,
    overdispersion = 2)

# Group sequential design alpha spending function design with O'Brien and
# Fleming type boundaries: Power and test characteristics for N = 264,
# under variable exposure time with uniform recruitment over 1.25 months,
# study time (accrual + followup) = 4, interim analysis take place after
# equidistant time points (lambda1, lambda2, and overdispersion as specified,
# no futility stopping):
dOF <- getDesignGroupSequential(
    kMax = 3,
    alpha = 0.025,
    beta = 0.2,
    typeOfDesign = "asOF")

getSimulationCounts(design = dOF,
    lambda1 = seq(0.04, 0.12, 0.02),
    lambda2 = 0.12,
    directionUpper = FALSE,
    overdispersion = 5,
    plannedCalendarTime = (1:3)/3*4,
    maxNumberOfSubjects = 264,
    followUpTime = 2.75,
    accrualTime = 1.25,
    maxNumberOfIterations = 1000)

## End(Not run)
```

`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 = NA_integer_,
  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. Must be created with getDesignInverseNormal(), getDesignFisher(), or getDesignFixed(); designs created with getDesignGroupSequential() or getDesignConditionalDunnett() are not supported.
...	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	Logical. 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 logical vector of length kMax - 1 indicating whether or not an adaptation takes place at interim k, default is rep(TRUE, kMax - 1).
typeOfSelection	Rule for selecting treatment arms (multi-arm designs) or populations (enrichment designs) for continuation after an interim analysis. The default "best" selects only the single best arm or population; "rBest" (also accepted as "rbest") selects the best rValue; "epsilon" selects those within epsilonValue of the best; "all" retains all remaining arms or populations; and "userDefined" applies selectArmsFunction or selectPopulationsFunction. Selection is applied at interims with adaptations = TRUE. See the section *Treatment arm and population selection* for details and an example of gradual elimination.
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. Must be a numeric of length 1.

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 numeric 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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.

minNumberOfSubjectsPerStage
: When performing a data driven sample size recalculation, the numeric 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 numeric 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 maxNumberOfSubjectsPe (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;

piTreatmentsH1 and piControlH1 in multi-arm designs, piTreatmentH1 and piControlH1 in enrichment designs), 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. Must be a numeric of length 1.

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. Must be a positive integer of length 1.

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 and specified minNumberOfSubjectsPerStage and maxNumberOfSubjectsPerSt (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 stage, conditionalPower, conditionalCriticalValue, plannedSubjects/planne allocationRatioPlanned, selectedPopulations, thetaH1 (for means and survival), stDevH1 (for means), overallEffects, and for rates additionally: piTreatmentsH1, piControlH1, overallRates, and overallRatesControl (see examples).

showStatistics Logical. 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.

Treatment arm and population selection

Selection determines which treatment arms or populations continue to the next stage. It is applied at interim analysis k only if adaptations[k] = TRUE; the default enables selection at every interim. Otherwise, the current selection is retained. Once dropped, an arm or population cannot be selected again. In multi-arm designs, these rules apply to the active treatment arms; the control group is retained.

The ranking uses the cumulative interim data and the criterion specified by effectMeasure: the estimated treatment effect for "effectEstimate" (the default), or the test statistic for "testStatistic". The selection functions use a vector effectVector on a scale where larger values rank better. For survival endpoints with effectMeasure = "effectEstimate", this is the estimated hazard ratio for directionUpper = TRUE and its reciprocal for directionUpper = FALSE.

"best" **(default)** Selects only the single best remaining arm or population. Thus, with three active arms and selection enabled at the first interim, only one arm continues to stage 2. It does not drop just the worst arm. If several have the same best value, the first in the original order is selected.

"rBest" Selects the rValue highest-ranked remaining arms or populations, subject to availability. Specify rValue as a positive integer. For example, rValue = 2 selects the best two; rValue = 1 gives the same selection as "best". The same rValue is used at every enabled interim, so this option does not mean dropping one arm or population at each interim. The spelling "rbest" is also accepted.

"epsilon" Selects the best and all other remaining arms or populations whose selection value is at most epsilonValue below the best value, including equality. Specify epsilonValue on the scale of effectVector. The number selected depends on the observed differences and can vary between simulation runs. For example, selection values c(0.20, 0.35, 0.40) and epsilonValue = 0.10 retain the second and third arms or populations. For response-rate differences, 0.10 represents 10 percentage points.

"all" Retains all remaining arms or populations without ranking them. This disables dropping based on relative performance, but threshold conditions and stopping rules still apply.

"userDefined" Uses selectArmsFunction in multi-arm designs or selectPopulationsFunction in enrichment designs. Supply a function returning a logical vector in the original arm or population order, with TRUE indicating selection. Its length must equal the original number of active arms or populations. The function can use stage, effectVector, and the other arguments listed in the corresponding parameter description. Use this option for rules whose number selected changes by stage, such as dropping the worst remaining arm at each interim.

For the four built-in rules, selected arms or populations must additionally have a selection value strictly above threshold. The default threshold = -Inf imposes no additional restriction. Consequently, fewer than the requested number may continue, or none may continue. For "userDefined", implement any threshold condition in the supplied function. Efficacy and futility stopping rules apply separately from selection.

For example, the following multi-arm selection function drops one worst remaining arm at each enabled interim, while retaining at least one arm. Pass it as `selectArmsFunction = dropWorstArm` together with `typeOfSelection = "userDefined"`. If three arms are still available at the first interim, it retains two; at the next enabled interim it retains one, provided the trial continues.

```
dropWorstArm <- function(effectVector, selectedArms, stage) {
    keep <- selectedArms[seq_along(effectVector), stage] & !is.na(effectVector)
    if (sum(keep) > 1L) {
        remaining <- which(keep)
        worst <- remaining[which.min(effectVector[remaining])]
        keep[worst] <- FALSE
    }
    keep
}
```

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

```
## Not run:
# 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
designIN <- 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
effectList <- list(subGroups=subGroups, prevalences=prevalences, stDevs = stDev, effects = effects)

# Perform simulation
simResultsPE <- getSimulationEnrichmentMeans(design = designIN,
    effectList = effectList, 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
designIN2 <- 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
effectList <- list(subGroups=subGroups, prevalences=prevalences, stDevs = stDev, effects = effects)

# Perform simulation
simResultsPE <- getSimulationEnrichmentMeans(
    design = designIN2,
    effectList = effectList,
    typeOfSelection = "userDefined",
    selectPopulationsFunction = mySelection,
    intersectionTest = "Simes",
    plannedSubjects = c(50, 100, 150),
    maxNumberOfIterations = 100)
print(simResultsPE)
if (require(ggplot2)) plot(simResultsPE, type = 3)

## End(Not run)
```

---

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 = NA,
  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 = NA_integer_,
  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. Must be created with `getDesignInverseNormal()`, `getDesignFisher()`, or `getDesignFixed()`; designs created with `getDesignGroupSequential()` or `getDesignConditionalDunnett()` are not supported.
`...`	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
: Logical. 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
: Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.

adaptations
: A logical vector of length kMax - 1 indicating whether or not an adaptation takes place at interim k, default is rep(TRUE, kMax - 1).

typeOfSelection
: Rule for selecting treatment arms (multi-arm designs) or populations (enrichment designs) for continuation after an interim analysis. The default "best" selects only the single best arm or population; "rBest" (also accepted as "rbest") selects the best rValue; "epsilon" selects those within epsilonValue of the best; "all" retains all remaining arms or populations; and "userDefined" applies selectArmsFunction or selectPopulationsFunction. Selection is applied at interims with adaptations = TRUE. See the section *Treatment arm and population selection* for details and an example of gradual elimination.

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. Must be a numeric of length 1.

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 numeric 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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length

kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.

minNumberOfSubjectsPerStage
: When performing a data driven sample size recalculation, the numeric 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 numeric 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 maxNumberOfSubjectsPe (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; piTreatmentsH1 and piControlH1 in multi-arm designs, piTreatmentH1 and piControlH1 in enrichment designs), 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. Must be a positive integer of length 1.

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 and specified minNumberOfSubjectsPerStage and maxNumberOfSubjectsPerSt (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 stage, conditionalPower, conditionalCriticalValue, plannedSubjects/plann allocationRatioPlanned, selectedPopulations, thetaH1 (for means and survival), stDevH1 (for means), overallEffects, and for rates additionally: piTreatmentsH1, piControlH1, overallRates, and overallRatesControl (see examples).

showStatistics Logical. 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.

Treatment arm and population selection

Selection determines which treatment arms or populations continue to the next stage. It is applied at interim analysis k only if adaptations[k] = TRUE; the default enables selection at every interim. Otherwise, the current selection is retained. Once dropped, an arm or population cannot be selected again. In multi-arm designs, these rules apply to the active treatment arms; the control group is retained.

The ranking uses the cumulative interim data and the criterion specified by effectMeasure: the estimated treatment effect for "effectEstimate" (the default), or the test statistic for "testStatistic". The selection functions use a vector effectVector on a scale where larger values rank better. For survival endpoints with effectMeasure = "effectEstimate", this is the estimated hazard ratio for directionUpper = TRUE and its reciprocal for directionUpper = FALSE.

"best" **(default)** Selects only the single best remaining arm or population. Thus, with three active arms and selection enabled at the first interim, only one arm continues to stage 2. It does not drop just the worst arm. If several have the same best value, the first in the original order is selected.

"rBest" Selects the rValue highest-ranked remaining arms or populations, subject to availability. Specify rValue as a positive integer. For example, rValue = 2 selects the best two; rValue = 1 gives the same selection as "best". The same rValue is used at every enabled interim, so this option does not mean dropping one arm or population at each interim. The spelling "rbest" is also accepted.

"epsilon" Selects the best and all other remaining arms or populations whose selection value is at most epsilonValue below the best value, including equality. Specify epsilonValue on the scale of effectVector. The number selected depends on the observed differences and can vary between simulation runs. For example, selection values c(0.20, 0.35, 0.40) and epsilonValue = 0.10 retain the second and third arms or populations. For response-rate differences, 0.10 represents 10 percentage points.

"all" Retains all remaining arms or populations without ranking them. This disables dropping based on relative performance, but threshold conditions and stopping rules still apply.

"userDefined" Uses selectArmsFunction in multi-arm designs or selectPopulationsFunction in enrichment designs. Supply a function returning a logical vector in the original arm or population order, with TRUE indicating selection. Its length must equal the original number of active arms or populations. The function can use stage, effectVector, and the other arguments listed in the corresponding parameter description. Use this option for rules whose number selected changes by stage, such as dropping the worst remaining arm at each interim.

For the four built-in rules, selected arms or populations must additionally have a selection value strictly above threshold. The default threshold = -Inf imposes no additional restriction. Consequently, fewer than the requested number may continue, or none may continue. For "userDefined", implement any threshold condition in the supplied function. Efficacy and futility stopping rules apply separately from selection.

For example, the following multi-arm selection function drops one worst remaining arm at each enabled interim, while retaining at least one arm. Pass it as selectArmsFunction = dropWorstArm together with typeOfSelection = "userDefined". If three arms are still available at the first interim, it retains two; at the next enabled interim it retains one, provided the trial continues.

```
dropWorstArm <- function(effectVector, selectedArms, stage) {
    keep <- selectedArms[seq_along(effectVector), stage] & !is.na(effectVector)
    if (sum(keep) > 1L) {
        remaining <- which(keep)
        worst <- remaining[which.min(effectVector[remaining])]
        keep[worst] <- FALSE
    }
    keep
}
```

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

```
## Not run:
# 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, ])
)
design <- getDesignInverseNormal(informationRates = c(0.4, 1),
    typeOfDesign = "noEarlyEfficacy")
simResultsPE <- getSimulationEnrichmentRates(design,
    plannedSubjects = c(150, 300),
    allocationRatioPlanned = 1.5, directionUpper = TRUE,
    effectList = effectList, stratifiedAnalysis = TRUE,
    intersectionTest = "Sidak",
    typeOfSelection = "epsilon", epsilonValue = 0.025,
    maxNumberOfIterations = 100)
print(simResultsPE)

## End(Not run)
```

`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.

Depending on `simulationType`, either a patient-wise survival simulation is performed or normally distributed log-rank test statistics are simulated. The default `simulationType = "auto"` chooses the simulation approach automatically based on the explicitly specified arguments. The null hypothesis is defined by the hazard ratio `thetaH0`; values other than 1 can be used, for example, to simulate non-inferiority designs.

The outputs are matrices with stage rows and scenario columns for two-arm trials. Multi-arm treatment outputs are arrays with dimensions stage, scenario, and active arm; shared control is a stage-by-scenario matrix. Enrichment outputs have dimensions stage, scenario, and population (subpopulations followed by the full population). A single active arm or population uses a stage-by-scenario matrix. Stages and scenarios are never dropped, including for a fixed design. Discontinued groups contribute through their discontinuation analysis and stopped trials through their stopping analysis, with no later contributions. Overlapping enrichment populations include their shared subjects.

Usage

```
getSimulationEnrichmentSurvival(
  design = NULL,
  ...,
  thetaH0 = 1,
```

```
  simulationType = c("auto", "patientWise", "testStatisticBased", "patientWiseBasic"),
  effectList = NULL,
  kappa = 1,
  eventTime = 12,
  accrualTime = c(0, 12),
  accrualIntensity = 0.1,
  accrualIntensityType = c("auto", "absolute", "relative"),
  dropoutRate1 = 0,
  dropoutRate2 = 0,
  dropoutTime = 12,
  maxNumberOfSubjects = NA_real_,
  intersectionTest = c("Simes", "SpiessensDebois", "Bonferroni", "Sidak"),
  stratifiedAnalysis = TRUE,
  directionUpper = NA,
  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 = NA_integer_,
  maxNumberOfRawDatasetsPerStage = 0,
  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. Must be created with getDesignInverseNormal(), getDesignFisher(), or getDesignFixed(); designs created with getDesignGroupSequential() or getDesignConditionalDunnett() are not supported.
...	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 general estimates, the normal case, and the binary case, 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 • *general estimates*: a value on the scale of the supplied estimates can be specified.

- *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.
- *count data*: a bound for testing H0: lambda1 / lambda2 = 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.

simulationType
: Character value specifying the simulation approach: "auto", "patientWise", "testStatisticBased", or "patientWiseBasic". If "auto" is specified, the simulation type is selected automatically based on the explicitly specified arguments. If patient-wise-specific arguments such as eventTime, accrualTime, accrualIntensity, dropoutRate1, dropoutRate2, dropoutTime, maxNumberOfSubjects, or kappa are provided, patient-wise simulation is used; otherwise the test-statistic-based simulation is used for backward compatibility.

effectList
: List of subsets, prevalences, and effect sizes with columns and number of rows reflecting the different situations to consider (see examples). For a piecewise exponential patient-wise survival simulation, specify subGroups, prevalences, and piecewiseSurvivalTime, where the latter is a PiecewiseSurvivalTimeEnrichment object created by getPiecewiseSurvivalTimeEnrichment(). Its subgroup names and the order of subGroups must agree. In this case, do not also specify piControls, piTreatments, or hazardRatios.

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(q = time, shape = kappa, scale = 1 / lambda) of the stats package.
For example, getPiecewiseExponentialDistribution(time = 130, piecewiseLambda = 0.01, kappa = 4.2) and pweibull(q = 130, shape = 4.2, scale = 1 / 0.01) provide the same result.

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 numeric 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. For multi-arm simulations, this can be a vector of length equal to the number of active arms, specifying different drop-out rates for each active arm.

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 are specified, this will be calculated. Must be a positive integer of length 1.

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
: Logical. 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
: Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.

adaptations
: A logical vector of length kMax - 1 indicating whether or not an adaptation takes place at interim k, default is rep(TRUE, kMax - 1).

typeOfSelection
: Rule for selecting treatment arms (multi-arm designs) or populations (enrichment designs) for continuation after an interim analysis. The default "best" selects only the single best arm or population; "rBest" (also accepted as "rbest") selects the best rValue; "epsilon" selects those within epsilonValue of the best; "all" retains all remaining arms or populations; and "userDefined" applies selectArmsFunction or selectPopulationsFunction. Selection is applied at interims with adaptations = TRUE. See the section *Treatment arm and population selection* for details and an example of gradual elimination.

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. Must be a numeric of length 1.

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 numeric 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 num-

ber 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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.

minNumberOfEventsPerStage
: When performing a data driven sample size recalculation, the numeric 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 numeric 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 maxNumberOfSubjectsPe (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; piTreatmentsH1 and piControlH1 in multi-arm designs, piTreatmentH1 and piControlH1 in enrichment designs), 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. Must be a numeric of length 1.

maxNumberOfIterations
: The number of simulation iterations, default is 1000. Must be a positive integer of length 1.

maxNumberOfRawDatasetsPerStage
: The maximum number of patient-level datasets to retain for each scenario and stopping stage. The default is 0, which retains no patient-level data. Use getRawData() to retrieve retained datasets.

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, event number recalculation is performed with conditional power and 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 stage, conditionalPower, conditionalCriticalValue, plannedSubjects/plann
allocationRatioPlanned, selectedPopulations, thetaH1 (for means and survival), stDevH1 (for means), overallEffects, and for rates additionally: piTreatmentsH1, piControlH1, overallRates, and overallRatesControl (see examples).

showStatistics Logical. 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.

If simulationType = "patientWise", patient-wise survival data are simulated based on the specified accrual, event-time, and dropout assumptions. If simulationType = "testStatisticBased", normally distributed log-rank test statistics are simulated instead. The default simulationType = "auto" selects the patient-wise approach if patient-wise-specific arguments were explicitly specified; otherwise the test-statistic-based approach is used for backward compatibility. After a population selection, each subsequent patient-wise analysis targets the planned increase in events among the selected subgroups. Events already observed in those subgroups at the preceding analysis count toward the target.

For all simulation types, populationEventsPerStage reports stage-wise event increments for populations still selected for analysis. Dropped populations contribute zero to these means, which are averaged over iterations reaching the stage. In contrast, singleEventsPerSubsetAndStage includes events in a subset whenever it belongs to a selected population, even if its own population hypothesis has been dropped.

Printed drop-out observation times are reported once per population because both treatment arms share the population's finishing analysis. The summary groups both arm-specific proportions and one observation time under each population. The arm-specific result fields remain available for extraction.

For piecewise exponential event times, include a PiecewiseSurvivalTimeEnrichment object in effectList$piecewiseSurvivalTime. Its hazard ratios are represented as a subgroup-by-interval-by-situation array; equivalently, they can be supplied to the constructor as a named list of subgroup-by-interval matrices. The situation dimension changes slowest, so printing the object displays one subgroup-by-interval matrix for each situation.

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, thetaH0, allocationRatioPlanned, minNumberOfEventsPerStage, maxNumberOfEventsPerStage, conditionalPower, conditionalCriticalValue, and overallEffects. The function has to contain the three-dots argument '...' (see examples).

The corresponding dropoutObservationTime1 and dropoutObservationTime2 fields retain the mean stage analysis time over exactly the same contributing replications, with the same dimensions as the proportions. They are omitted from print and summary output but can be accessed directly as result$dropoutObservationTime1 and result$dropoutObservationTime2. In multi-arm trials, the first field refers to the active treatment arms and the second to shared control. In enrichment

trials, both fields contain population-specific results for their respective treatment arms. See SimulationResultsSurvival, SimulationResultsMultiArmSurvival, and SimulationResultsEnrichmentSurvival for the result fields and their dimensions.

These averages can differ from the trial-level `analysisTime`, because arm or population selection and the requirement for a nonempty enrolled-subject denominator can change which replications contribute. Within a replication, all arms or populations contributing at a given stage share its analysis time; differences between the reported averages arise from the contributing sets of replications, rather than different analysis schedules within that replication. When both input dropout rates are zero, the proportions are also omitted from print and summary output while remaining in the result object.

These outputs require patient histories and are unavailable for `simulationType = "testStatisticBased"`. Existing recruitment and allocation mechanisms are retained. The input dropout probability at `dropoutTime` describes the dropout distribution without competing events; it is not the simulated observed proportion. Analytic expected proportions condition on deterministic planned accrual at each planned analysis time, whereas simulation includes random analysis times, early stopping and selection. Compare them only with matching stage, horizon, enrollment and allocation assumptions.

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`.

Treatment arm and population selection

Selection determines which treatment arms or populations continue to the next stage. It is applied at interim analysis `k` only if `adaptations[k] = TRUE`; the default enables selection at every interim. Otherwise, the current selection is retained. Once dropped, an arm or population cannot be selected again. In multi-arm designs, these rules apply to the active treatment arms; the control group is retained.

The ranking uses the cumulative interim data and the criterion specified by `effectMeasure`: the estimated treatment effect for `"effectEstimate"` (the default), or the test statistic for `"testStatistic"`. The selection functions use a vector `effectVector` on a scale where larger values rank better. For survival endpoints with `effectMeasure = "effectEstimate"`, this is the estimated hazard ratio for `directionUpper = TRUE` and its reciprocal for `directionUpper = FALSE`.

`"best"` **(default)** Selects only the single best remaining arm or population. Thus, with three active arms and selection enabled at the first interim, only one arm continues to stage 2. It does not drop just the worst arm. If several have the same best value, the first in the original order is selected.

`"rBest"` Selects the `rValue` highest-ranked remaining arms or populations, subject to availability. Specify `rValue` as a positive integer. For example, `rValue = 2` selects the best two; `rValue = 1` gives the same selection as `"best"`. The same `rValue` is used at every enabled interim, so this option does not mean dropping one arm or population at each interim. The spelling `"rbest"` is also accepted.

`"epsilon"` Selects the best and all other remaining arms or populations whose selection value is at most `epsilonValue` below the best value, including equality. Specify `epsilonValue` on the scale of `effectVector`. The number selected depends on the observed differences and can vary between simulation runs. For example, selection values `c(0.20, 0.35, 0.40)` and `epsilonValue = 0.10` retain the second and third arms or populations. For response-rate differences, `0.10` represents 10 percentage points.

`"all"` Retains all remaining arms or populations without ranking them. This disables dropping based on relative performance, but threshold conditions and stopping rules still apply.

`"userDefined"` Uses `selectArmsFunction` in multi-arm designs or `selectPopulationsFunction` in enrichment designs. Supply a function returning a logical vector in the original arm or population order, with `TRUE` indicating selection. Its length must equal the original number of active arms or populations. The function can use `stage`, `effectVector`, and the other arguments listed in the corresponding parameter description. Use this option for rules whose number selected changes by stage, such as dropping the worst remaining arm at each interim.

For the four built-in rules, selected arms or populations must additionally have a selection value strictly above `threshold`. The default `threshold = -Inf` imposes no additional restriction. Consequently, fewer than the requested number may continue, or none may continue. For `"userDefined"`, implement any threshold condition in the supplied function. Efficacy and futility stopping rules apply separately from selection.

For example, the following multi-arm selection function drops one worst remaining arm at each enabled interim, while retaining at least one arm. Pass it as `selectArmsFunction = dropWorstArm` together with `typeOfSelection = "userDefined"`. If three arms are still available at the first interim, it retains two; at the next enabled interim it retains one, provided the trial continues.

```
dropWorstArm <- function(effectVector, selectedArms, stage) {
    keep <- selectedArms[seq_along(effectVector), stage] & !is.na(effectVector)
    if (sum(keep) > 1L) {
        remaining <- which(keep)
        worst <- remaining[which.min(effectVector[remaining])]
        keep[worst] <- FALSE
    }
    keep
}
```

Conditional power under piecewise hazards

Conditional power is calculated using a scalar hazard ratio estimated from the interim data or specified through `thetaH1`. The calculation uses a proportional-hazards working model, including when survival times are generated from piecewise hazards with interval-specific hazard ratios. The interval-specific hazards are not used to project future test-statistic drift in this calculation.

Accordingly, `conditionalPowerAchieved` reports conditional power under the scalar effect assumption for the selected number of events. It is not an empirical rejection probability and, when hazard ratios vary over time, need not equal the conditional rejection probability under the specified piecewise survival model. The default event-number recalculation uses the same working model; a custom `calcEventsFunction` can define a different recalculation rule.

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

```
## Not run:
# 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)

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

simResultsPE <- getSimulationEnrichmentSurvival(design,
    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)

# Non-inferiority in the full population (H0: hazard ratio = 1.2)
effectListNonInferiority <- list(
    subGroups = "F", prevalences = 1,
    hazardRatios = matrix(1), piControls = 0.3)
simulationResultsNonInferiority <- getSimulationEnrichmentSurvival(
    design = getDesignFixed(sided = 1), thetaH0 = 1.2,
    directionUpper = FALSE, effectList = effectListNonInferiority,
    plannedEvents = 80, allocationRatioPlanned = 1,
    maxNumberOfSubjects = 400, accrualTime = c(0, 20),
    accrualIntensity = 20, maxNumberOfIterations = 50,
    simulationType = "patientWise")

# Piecewise exponential event times in two disjoint subgroups
piecewiseTime <- getPiecewiseSurvivalTimeEnrichment(
    piecewiseSurvivalTime = c(0, 6),
```

```
    lambdaControls = rbind(S = c(0.08, 0.12), R = c(0.05, 0.09)),
    hazardRatio = list(
        favorable = rbind(S = c(0.65, 0.55), R = c(0.9, 0.8)),
        conservative = rbind(S = c(0.8, 0.7), R = c(1, 0.9))
    )
)
effectListPiecewise <- list(
    subGroups = c("S", "R"), prevalences = c(0.4, 0.6),
    piecewiseSurvivalTime = piecewiseTime)
simulationResultsPiecewise <- getSimulationEnrichmentSurvival(
    design = getDesignFixed(), simulationType = "patientWise",
    effectList = effectListPiecewise, directionUpper = FALSE,
    plannedEvents = 80, maxNumberOfSubjects = 300,
    accrualTime = c(0, 12), accrualIntensity = 25,
    maxNumberOfIterations = 50)

## End(Not run)
```

`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 = NA,
  allocationRatioPlanned = NA_real_,
  minNumberOfSubjectsPerStage = NA_real_,
  maxNumberOfSubjectsPerStage = NA_real_,
  conditionalPower = NA_real_,
  thetaH1 = NA_real_,
  stDevH1 = NA_real_,
  maxNumberOfIterations = NA_integer_,
  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 general estimates, the normal case, and the binary case, 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

- *general estimates*: a value on the scale of the supplied estimates can be specified.
- *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.
- *count data*: a bound for testing H0: `lambda1 / lambda2 = 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`. For two-armed trials, it is allowed to specify the standard deviations separately, i.e., as vector with two elements. If `meanRatio = TRUE` is specified, `stDev` defines the coefficient of variation `sigma / mu2`.

`plannedSubjects`
`plannedSubjects` is a numeric 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` Logical. Specifies the direction of the alternative, only applicable for one-sided testing; `TRUE` means that larger values of the test statistics yield smaller p-values. The default is `TRUE` for means, rates, and count data, and `FALSE` for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified `directionUpper` must agree with it.

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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.

minNumberOfSubjectsPerStage
: When performing a data driven sample size recalculation, the numeric 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 numeric 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 maxNumberOfSubjectsPe (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; piTreatmentsH1 and piControlH1 in multi-arm designs, piTreatmentH1 and piControlH1 in enrichment designs), 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. Must be a numeric of length 1.

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. Must be a positive integer of length 1.

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 and specified minNumberOfSubjectsPerStage and maxNumberOfSubjectsPerSt (see details and examples).

showStatistics
: Logical. 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`, `conditionalCriticalValu` `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

```
## Not run:
# 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,
        allocationRatioPlanned,
        thetaH1,
        stDevH1) {
    if (stage <= 2) {
        # Note that allocationRatioPlanned is as a vector of length kMax
        stageSubjects <- (1 + allocationRatioPlanned[stage])^2 /
            allocationRatioPlanned[stage] *
            (max(0, conditionalCriticalValue + stats::qnorm(conditionalPower)))^2 /
            (max(1e-12, thetaH1 / stDevH1))^2
        stageSubjects <- min(max(
            minNumberOfSubjectsPerStage[stage],
            stageSubjects
        ), maxNumberOfSubjectsPerStage[stage])
    } else {
        stageSubjects <- sampleSizesPerStage[stage - 1]
    }
    return(stageSubjects)
}
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,
    calcSubjectsFunction = mySampleSizeCalculationFunction,
    maxNumberOfIterations = 50
)

## End(Not run)
```

---

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 = NA_integer_,
  effectMatrix = NULL,
  typeOfShape = c("linear", "sigmoidEmax", "userDefined"),
  muMaxVector = seq(0, 1, 0.2),
  gED50 = NA_real_,
  slope = 1,
  doseLevels = NA_real_,
 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_,
  thetaH0 = C_THETA_H0_MEANS_DEFAULT,
  thetaH1 = NA_real_,
  stDevH1 = NA_real_,
  maxNumberOfIterations = NA_integer_,
  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. Must be created with getDesignInverseNormal(), getDesignFisher(), getDesignConditionalDunnett(), or getDesignFixed(); designs created with getDesignGroupSequential() are not supported.
...	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. Must not exceed 8.

effectMatrix
: Matrix of effect sizes with activeArms columns and one or more rows, where each row represents a different situation to consider.

typeOfShape
: The shape of the dose-response relationship over the treatment groups. This can be either "linear", "sigmoidEmax", or "userDefined", default is "linear". For "linear", muMaxVector specifies the range of effect sizes for the treatment group with highest response. If "sigmoidEmax" is selected, gED50 and slope has to be entered to specify the ED50 and the slope of the sigmoid Emax model. For "sigmoidEmax", muMaxVector specifies the range of effect sizes for the treatment group with response according to infinite dose. 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.

doseLevels
: The dose levels for the dose response relationship. If not specified, these dose levels are 1, ..., activeArms.

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. For two-armed trials, it is allowed to specify the standard deviations separately, i.e., as vector with two elements. If meanRatio = TRUE is specified, stDev defines the coefficient of variation sigma / mu2.

adaptations
: A logical vector of length kMax - 1 indicating whether or not an adaptation takes place at interim k, default is rep(TRUE, kMax - 1).

typeOfSelection
: Rule for selecting treatment arms (multi-arm designs) or populations (enrichment designs) for continuation after an interim analysis. The default "best" selects only the single best arm or population; "rBest" (also accepted as "rbest") selects the best rValue; "epsilon" selects those within epsilonValue of the best; "all" retains all remaining arms or populations; and "userDefined" applies selectArmsFunction or selectPopulationsFunction. Selection is applied at interims with adaptations = TRUE. See the section *Treatment arm and population selection* for details and an example of gradual elimination.

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. Must be a numeric of length 1.

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 numeric 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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.

minNumberOfSubjectsPerStage
: When performing a data driven sample size recalculation, the numeric 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 numeric 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 maxNumberOfSubjectsPe (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; piTreatmentsH1 and piControlH1 in multi-arm designs, piTreatmentH1 and piControlH1 in enrichment designs), or thetaH1 (for simulating hazard ratios) as parameters under which it is calculated and the sample size recalculation is performed.

thetaH0
: The null hypothesis value, default is 0 for general estimates, the normal case, and the binary case, 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

- *general estimates*: a value on the scale of the supplied estimates can be specified.
- *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.
- *count data*: a bound for testing H0: `lambda1 / lambda2 = 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`.

`thetaH1` If specified, the value of the alternative under which the conditional power or sample size recalculation calculation is performed. Must be a numeric of length 1.

`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`. Must be a positive integer of length 1.

`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 and specified `minNumberOfSubjectsPerStage` and `maxNumberOfSubjectsPerSt` (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`, `stage`, `conditionalPower`, `conditionalCriticalValue`, `plannedSubjects/planne` `allocationRatioPlanned`, `selectedArms`, `thetaH1` (for means and survival), `stDevH1` (for means), `overallEffects`, and for rates additionally: `piTreatmentsH1`, `piControlH1`, `overallRates`, and `overallRatesControl` (see examples).

`showStatistics` Logical. 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.

Treatment arm and population selection

Selection determines which treatment arms or populations continue to the next stage. It is applied at interim analysis k only if adaptations[k] = TRUE; the default enables selection at every interim. Otherwise, the current selection is retained. Once dropped, an arm or population cannot be selected again. In multi-arm designs, these rules apply to the active treatment arms; the control group is retained.

The ranking uses the cumulative interim data and the criterion specified by effectMeasure: the estimated treatment effect for "effectEstimate" (the default), or the test statistic for "testStatistic". The selection functions use a vector effectVector on a scale where larger values rank better. For survival endpoints with effectMeasure = "effectEstimate", this is the estimated hazard ratio for directionUpper = TRUE and its reciprocal for directionUpper = FALSE.

"best" **(default)** Selects only the single best remaining arm or population. Thus, with three active arms and selection enabled at the first interim, only one arm continues to stage 2. It does not drop just the worst arm. If several have the same best value, the first in the original order is selected.

"rBest" Selects the rValue highest-ranked remaining arms or populations, subject to availability. Specify rValue as a positive integer. For example, rValue = 2 selects the best two; rValue = 1 gives the same selection as "best". The same rValue is used at every enabled interim, so this option does not mean dropping one arm or population at each interim. The spelling "rbest" is also accepted.

"epsilon" Selects the best and all other remaining arms or populations whose selection value is at most epsilonValue below the best value, including equality. Specify epsilonValue on the scale of effectVector. The number selected depends on the observed differences and can vary between simulation runs. For example, selection values c(0.20, 0.35, 0.40) and epsilonValue = 0.10 retain the second and third arms or populations. For response-rate differences, 0.10 represents 10 percentage points.

"all" Retains all remaining arms or populations without ranking them. This disables dropping based on relative performance, but threshold conditions and stopping rules still apply.

"userDefined" Uses selectArmsFunction in multi-arm designs or selectPopulationsFunction in enrichment designs. Supply a function returning a logical vector in the original arm or population order, with TRUE indicating selection. Its length must equal the original number of active arms or populations. The function can use stage, effectVector, and the other arguments listed in the corresponding parameter description. Use this option for rules whose number selected changes by stage, such as dropping the worst remaining arm at each interim.

For the four built-in rules, selected arms or populations must additionally have a selection value strictly above `threshold`. The default `threshold = -Inf` imposes no additional restriction. Consequently, fewer than the requested number may continue, or none may continue. For `"userDefined"`, implement any threshold condition in the supplied function. Efficacy and futility stopping rules apply separately from selection.

For example, the following multi-arm selection function drops one worst remaining arm at each enabled interim, while retaining at least one arm. Pass it as `selectArmsFunction = dropWorstArm` together with `typeOfSelection = "userDefined"`. If three arms are still available at the first interim, it retains two; at the next enabled interim it retains one, provided the trial continues.

```
dropWorstArm <- function(effectVector, selectedArms, stage) {
    keep <- selectedArms[seq_along(effectVector), stage] & !is.na(effectVector)
    if (sum(keep) > 1L) {
        remaining <- which(keep)
        worst <- remaining[which.min(effectVector[remaining])]
        keep[worst] <- FALSE
    }
    keep
}
```

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

```
## Not run:
# 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)

## End(Not run)
```

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 = NA_integer_,
  effectMatrix = NULL,
  typeOfShape = c("linear", "sigmoidEmax", "userDefined"),
  piMaxVector = NA_real_,
  piControl = NA_real_,
  gED50 = NA_real_,
  slope = 1,
  doseLevels = NA_real_,
 intersectionTest = c("Dunnett", "Bonferroni", "Simes", "Sidak", "Hierarchical"),
  directionUpper = NA,
  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_,
  thetaH0 = C_THETA_H0_MEANS_DEFAULT,
  piTreatmentsH1 = NA_real_,
  piControlH1 = NA_real_,
  maxNumberOfIterations = NA_integer_,
  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. Must be created with getDesignInverseNormal(), getDesignFisher(), getDesignConditionalDunnett(), or getDesignFixed(); designs created with getDesignGroupSequential() are not supported.
...	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. Must not exceed 8.
effectMatrix	Matrix of effect sizes with activeArms columns and one or more rows, where each row represents a different situation to consider.
typeOfShape	The shape of the dose-response relationship over the treatment groups. This can be either "linear", "sigmoidEmax", or "userDefined", default is "linear". For "linear", piMaxVector specifies the range of effect sizes for the treatment group with highest response. If "sigmoidEmax" is selected, gED50 and slope has to be entered to specify the ED50 and the slope of the sigmoid Emax model. For "sigmoidEmax", piMaxVector specifies the range of effect sizes for the treatment group with response according to infinite dose. 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, default is 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.
doseLevels	The dose levels for the dose response relationship. If not specified, these dose levels are 1,...,activeArms.
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	Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.
adaptations	A logical vector of length kMax - 1 indicating whether or not an adaptation takes place at interim k, default is rep(TRUE, kMax - 1).
typeOfSelection	Rule for selecting treatment arms (multi-arm designs) or populations (enrichment designs) for continuation after an interim analysis. The default "best" selects only the single best arm or population; "rBest" (also accepted as "rbest") selects the best rValue; "epsilon" selects those within epsilonValue of the

best; "all" retains all remaining arms or populations; and "userDefined" applies selectArmsFunction or selectPopulationsFunction. Selection is applied at interims with adaptations = TRUE. See the section *Treatment arm and population selection* for details and an example of gradual elimination.

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. Must be a numeric of length 1.

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 numeric 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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.

minNumberOfSubjectsPerStage
: When performing a data driven sample size recalculation, the numeric 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 numeric 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 maxNumberOfSubjectsPe (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; piTreatmentsH1 and piControlH1 in multi-arm designs, piTreatmentH1 and piControlH1 in enrichment designs), or thetaH1 (for simulating hazard ratios) as parameters under which it is calculated and the sample size recalculation is performed.

thetaH0
: The null hypothesis value, default is 0 for general estimates, the normal case, and the binary case, 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

  - *general estimates*: a value on the scale of the supplied estimates can be specified.
  - *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.
  - *count data*: a bound for testing H0: lambda1 / lambda2 = 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.

piTreatmentsH1
: 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. Must be a positive integer of length 1.

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 and specified minNumberOfSubjectsPerStage and maxNumberOfSubjectsPerSt (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, stage, conditionalPower, conditionalCriticalValue, plannedSubjects/planne allocationRatioPlanned, selectedArms, thetaH1 (for means and survival), stDevH1 (for means), overallEffects, and for rates additionally: piTreatmentsH1, piControlH1, overallRates, and overallRatesControl (see examples).

showStatistics
: Logical. 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 `piTreatmentsH1` 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`, `selectedArms`, `directionUpper`, `plannedSubjects`, `allocationRatioPlanned`, `minNumberOfSubjectsPerStage`, `maxNumberOfSubjectsPerStage`, `conditionalPower`, `conditionalCriticalValue`, `overallRates`, `overallRatesControl`, `piTreatmentsH1`, 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`.

Treatment arm and population selection

Selection determines which treatment arms or populations continue to the next stage. It is applied at interim analysis `k` only if `adaptations[k] = TRUE`; the default enables selection at every interim. Otherwise, the current selection is retained. Once dropped, an arm or population cannot be selected again. In multi-arm designs, these rules apply to the active treatment arms; the control group is retained.

The ranking uses the cumulative interim data and the criterion specified by `effectMeasure`: the estimated treatment effect for `"effectEstimate"` (the default), or the test statistic for `"testStatistic"`. The selection functions use a vector `effectVector` on a scale where larger values rank better. For survival endpoints with `effectMeasure = "effectEstimate"`, this is the estimated hazard ratio for `directionUpper = TRUE` and its reciprocal for `directionUpper = FALSE`.

`"best"` **(default)** Selects only the single best remaining arm or population. Thus, with three active arms and selection enabled at the first interim, only one arm continues to stage 2. It does not drop just the worst arm. If several have the same best value, the first in the original order is selected.

`"rBest"` Selects the `rValue` highest-ranked remaining arms or populations, subject to availability. Specify `rValue` as a positive integer. For example, `rValue = 2` selects the best two; `rValue = 1` gives the same selection as `"best"`. The same `rValue` is used at every enabled interim, so this option does not mean dropping one arm or population at each interim. The spelling `"rbest"` is also accepted.

`"epsilon"` Selects the best and all other remaining arms or populations whose selection value is at most `epsilonValue` below the best value, including equality. Specify `epsilonValue` on the scale of `effectVector`. The number selected depends on the observed differences and can vary between simulation runs. For example, selection values `c(0.20, 0.35, 0.40)` and `epsilonValue = 0.10` retain the second and third arms or populations. For response-rate differences, `0.10` represents 10 percentage points.

`"all"` Retains all remaining arms or populations without ranking them. This disables dropping based on relative performance, but threshold conditions and stopping rules still apply.

`"userDefined"` Uses `selectArmsFunction` in multi-arm designs or `selectPopulationsFunction` in enrichment designs. Supply a function returning a logical vector in the original arm or population order, with `TRUE` indicating selection. Its length must equal the original number of active arms or populations. The function can use `stage`, `effectVector`, and the other arguments listed in the corresponding parameter description. Use this option for rules whose number selected changes by stage, such as dropping the worst remaining arm at each interim.

For the four built-in rules, selected arms or populations must additionally have a selection value strictly above `threshold`. The default `threshold = -Inf` imposes no additional restriction. Consequently, fewer than the requested number may continue, or none may continue. For `"userDefined"`, implement any threshold condition in the supplied function. Efficacy and futility stopping rules apply separately from selection.

For example, the following multi-arm selection function drops one worst remaining arm at each enabled interim, while retaining at least one arm. Pass it as `selectArmsFunction = dropWorstArm` together with `typeOfSelection = "userDefined"`. If three arms are still available at the first interim, it retains two; at the next enabled interim it retains one, provided the trial continues.

```
dropWorstArm <- function(effectVector, selectedArms, stage) {
    keep <- selectedArms[seq_along(effectVector), stage] & !is.na(effectVector)
    if (sum(keep) > 1L) {
        remaining <- which(keep)
        worst <- remaining[which.min(effectVector[remaining])]
        keep[worst] <- FALSE
    }
    keep
}
```

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

```
## Not run:
# 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.
design <- 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 = design, typeOfShape = "userDefined",
    effectMatrix = effectMatrix , piControl = 0.2,
    typeOfSelection = "best", threshold = 0, intersectionTest = "Dunnett",
    plannedSubjects = c(20, 40, 60),
    maxNumberOfIterations = 50)
summary(x)

## End(Not run)
```

`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.

Depending on `simulationType`, either a patient-wise survival simulation is performed, a patient-wise basic approximation is used, or normally distributed log-rank test statistics are simulated. The default `simulationType = "auto"` chooses the simulation approach automatically based on the explicitly specified arguments. The null hypothesis is defined by the hazard ratio `thetaH0`; values other than 1 can be used, for example, to simulate non-inferiority designs.

The outputs are matrices with stage rows and scenario columns for two-arm trials. Multi-arm treatment outputs are arrays with dimensions stage, scenario, and active arm; shared control is a stage-by-scenario matrix. Enrichment outputs have dimensions stage, scenario, and population (subpopulations followed by the full population). A single active arm or population uses a stage-by-scenario matrix. Stages and scenarios are never dropped, including for a fixed design. Discontinued groups contribute through their discontinuation analysis and stopped trials through their stopping analysis, with no later contributions. Overlapping enrichment populations include their shared subjects.

Usage

```
getSimulationMultiArmSurvival(
  design = NULL,
  ...,
 simulationType = c("auto", "patientWise", "testStatisticBased", "patientWiseBasic"),
  thetaH0 = 1,
  activeArms = NA_integer_,
  piControl = NA_real_,
  piecewiseSurvivalTime = NULL,
  effectMatrix = NULL,
  typeOfShape = c("linear", "sigmoidEmax", "userDefined"),
  omegaMaxVector = NA_real_,
  kappa = 1,
  gED50 = NA_real_,
  slope = 1,
```

```
  doseLevels = NA_real_,
  eventTime = 12,
  accrualTime = c(0, 12),
  accrualIntensity = 0.1,
  accrualIntensityType = c("auto", "absolute", "relative"),
  dropoutRate1 = 0,
  dropoutRate2 = 0,
  dropoutTime = 12,
  maxNumberOfSubjects = NA_real_,
 intersectionTest = c("Dunnett", "Bonferroni", "Simes", "Sidak", "Hierarchical"),
  directionUpper = NA,
  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 = NA_integer_,
  maxNumberOfRawDatasetsPerStage = 0,
  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. Must be created with getDesignInverseNormal(), getDesignFisher(), getDesignConditionalDunnett(), or getDesignFixed(); designs created with getDesignGroupSequential() are not supported.
...	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
simulationType	Character value specifying the simulation approach: "auto", "patientWise", "testStatisticBased", or "patientWiseBasic". If "auto" is specified, the simulation type is selected automatically based on the explicitly specified arguments. If patient-wise-specific arguments such as piControl, eventTime, accrualTime, accrualIntensity, accrualIntensityType, dropoutRate1, dropoutRate2, dropoutTime, maxNumberOfSubjects, or kappa are provided, a patient-wise simulation approach is used; otherwise the test-statistic-based simulation is used for backward compatibility.
thetaH0	The null hypothesis value, default is 0 for general estimates, the normal case,

and the binary case, 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

- *general estimates*: a value on the scale of the supplied estimates can be specified.
- *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.
- *count data*: a bound for testing H0: lambda1 / lambda2 = 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.

activeArms — The number of active treatment arms to be compared with control, default is 3. Must not exceed 8.

piControl — The assumed event probability in the control arm by eventTime, default is 0.5. If specified and simulationType = "auto", the patient-wise simulation is used, which requires maxNumberOfSubjects or an absolute accrualIntensity.

piecewiseSurvivalTime — A PiecewiseSurvivalTimeMultiArm object created by getPiecewiseSurvivalTimeMultiArm(). It specifies common interval start times, the control hazard in every interval, and an active-arm-by-interval hazard-ratio matrix for every situation. It can only be used for patient-wise simulation with kappa = 1, and cannot be combined with piControl or effectMatrix.

effectMatrix — Matrix of effect sizes with activeArms columns and one or more rows, where each row represents a different situation to consider.

typeOfShape — The shape of the dose-response relationship over the treatment groups. This can be either "linear", "sigmoidEmax", or "userDefined", default is "linear". For "linear", omegaMaxVector specifies the range of effect sizes for the treatment group with highest response. If "sigmoidEmax" is selected, gED50 and slope has to be entered to specify the ED50 and the slope of the sigmoid Emax model. For "sigmoidEmax", omegaMaxVector specifies the range of effect sizes for the treatment group with response according to infinite dose. 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, 0.4, -0.2).

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(q = time, shape = kappa, scale = 1 / lambda) of the stats package.
For example, getPiecewiseExponentialDistribution(time = 130, piecewiseLambda = 0.01, kappa = 4.2) and pweibull(q = 130, shape = 4.2, scale = 1 / 0.01) provide the same result.

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.

doseLevels The dose levels for the dose response relationship. If not specified, these dose levels are 1,...,activeArms.

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 numeric 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. For multi-arm simulations, this can be a vector of length equal to the number of active arms, specifying different drop-out rates for each active arm.

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 are specified, this will be calculated. Must be a positive integer of length 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 Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.

adaptations A logical vector of length kMax - 1 indicating whether or not an adaptation takes place at interim k, default is rep(TRUE, kMax - 1).

typeOfSelection
Rule for selecting treatment arms (multi-arm designs) or populations (enrichment designs) for continuation after an interim analysis. The default "best" selects only the single best arm or population; "rBest" (also accepted as "rbest") selects the best rValue; "epsilon" selects those within epsilonValue of the best; "all" retains all remaining arms or populations; and "userDefined" applies selectArmsFunction or selectPopulationsFunction. Selection is applied at interims with adaptations = TRUE. See the section *Treatment arm and population selection* for details and an example of gradual elimination.

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. Must be a numeric of length 1.

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 numeric 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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.

minNumberOfEventsPerStage
: When performing a data driven sample size recalculation, the numeric 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 numeric 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 maxNumberOfSubjectsPe (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; piTreatmentsH1 and piControlH1 in multi-arm designs, piTreatmentH1 and piControlH1 in enrichment designs), 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. Must be a numeric of length 1.

maxNumberOfIterations
The number of simulation iterations, default is 1000. Must be a positive integer of length 1.

maxNumberOfRawDatasetsPerStage
The maximum number of patient-level datasets to retain for each scenario and stopping stage. The default is 0, which retains no patient-level data. Use getRawData() to retrieve retained datasets.

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, event number recalculation is performed with conditional power and 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, stage, conditionalPower, conditionalCriticalValue, plannedSubjects/planne allocationRatioPlanned, selectedArms, thetaH1 (for means and survival), stDevH1 (for means), overallEffects, and for rates additionally: piTreatmentsH1, piControlH1, overallRates, and overallRatesControl (see examples).

showStatistics Logical. 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 analysis times, power, stopping probabilities, expected sample size and events, and selection probabilities, at given number of subjects, events, parameter configuration, and treatment arm selection rule in the multi-arm survival design 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.

If simulationType = "patientWise", patient-wise survival data are simulated based on the specified accrual, event-time, and dropout assumptions. If simulationType = "patientWiseBasic", a simplified patient-wise survival approach is used. If simulationType = "testStatisticBased", normally distributed log-rank test statistics are simulated instead. The default simulationType = "auto" selects the simulation approach automatically based on the explicitly specified arguments and uses the legacy test-statistic-based approach for backward compatibility if no patient-wise-specific arguments are provided.

The summary groups observed drop-out proportions and observation times by active treatment arm, followed by the shared control arm. Each arm uses its own finishing analysis, including active arms discontinued at an interim.

For piecewise exponential event times, create piecewiseSurvivalTime with getPiecewiseSurvivalTimeMultiArm(). Hazard ratios are represented as an active-arm-by-interval-by-situation array; equivalently, they can

be supplied to the constructor as a named list of active-arm-by-interval matrices. The situation dimension changes slowest, so printing the object displays one active-arm-by-interval matrix for each situation.

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, thetaH0, allocationRatioPlanned, minNumberOfEventsPerStage, maxNumberOfEventsPerStage, conditionalPower, conditionalCriticalValue, and overallEffects. The function has to contain the three-dots argument '...' (see examples).

The corresponding dropoutObservationTime1 and dropoutObservationTime2 fields retain the mean stage analysis time over exactly the same contributing replications, with the same dimensions as the proportions. They are omitted from print and summary output but can be accessed directly as result$dropoutObservationTime1 and result$dropoutObservationTime2. In multi-arm trials, the first field refers to the active treatment arms and the second to shared control. In enrichment trials, both fields contain population-specific results for their respective treatment arms. See SimulationResultsSurvival, SimulationResultsMultiArmSurvival, and SimulationResultsEnrichmentSurvival for the result fields and their dimensions.

These averages can differ from the trial-level analysisTime, because arm or population selection and the requirement for a nonempty enrolled-subject denominator can change which replications contribute. Within a replication, all arms or populations contributing at a given stage share its analysis time; differences between the reported averages arise from the contributing sets of replications, rather than different analysis schedules within that replication. When both input dropout rates are zero, the proportions are also omitted from print and summary output while remaining in the result object.

These outputs require patient histories and are unavailable for simulationType = "testStatisticBased". Existing recruitment and allocation mechanisms are retained. The input dropout probability at dropoutTime describes the dropout distribution without competing events; it is not the simulated observed proportion. Analytic expected proportions condition on deterministic planned accrual at each planned analysis time, whereas simulation includes random analysis times, early stopping and selection. Compare them only with matching stage, horizon, enrollment and allocation assumptions.

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.

Treatment arm and population selection

Selection determines which treatment arms or populations continue to the next stage. It is applied at interim analysis k only if adaptations[k] = TRUE; the default enables selection at every interim. Otherwise, the current selection is retained. Once dropped, an arm or population cannot be selected

again. In multi-arm designs, these rules apply to the active treatment arms; the control group is retained.

The ranking uses the cumulative interim data and the criterion specified by `effectMeasure`: the estimated treatment effect for `"effectEstimate"` (the default), or the test statistic for `"testStatistic"`. The selection functions use a vector `effectVector` on a scale where larger values rank better. For survival endpoints with `effectMeasure = "effectEstimate"`, this is the estimated hazard ratio for `directionUpper = TRUE` and its reciprocal for `directionUpper = FALSE`.

`"best"` **(default)** Selects only the single best remaining arm or population. Thus, with three active arms and selection enabled at the first interim, only one arm continues to stage 2. It does not drop just the worst arm. If several have the same best value, the first in the original order is selected.

`"rBest"` Selects the `rValue` highest-ranked remaining arms or populations, subject to availability. Specify `rValue` as a positive integer. For example, `rValue = 2` selects the best two; `rValue = 1` gives the same selection as `"best"`. The same `rValue` is used at every enabled interim, so this option does not mean dropping one arm or population at each interim. The spelling `"rbest"` is also accepted.

`"epsilon"` Selects the best and all other remaining arms or populations whose selection value is at most `epsilonValue` below the best value, including equality. Specify `epsilonValue` on the scale of `effectVector`. The number selected depends on the observed differences and can vary between simulation runs. For example, selection values `c(0.20, 0.35, 0.40)` and `epsilonValue = 0.10` retain the second and third arms or populations. For response-rate differences, `0.10` represents 10 percentage points.

`"all"` Retains all remaining arms or populations without ranking them. This disables dropping based on relative performance, but threshold conditions and stopping rules still apply.

`"userDefined"` Uses `selectArmsFunction` in multi-arm designs or `selectPopulationsFunction` in enrichment designs. Supply a function returning a logical vector in the original arm or population order, with `TRUE` indicating selection. Its length must equal the original number of active arms or populations. The function can use `stage`, `effectVector`, and the other arguments listed in the corresponding parameter description. Use this option for rules whose number selected changes by stage, such as dropping the worst remaining arm at each interim.

For the four built-in rules, selected arms or populations must additionally have a selection value strictly above `threshold`. The default `threshold = -Inf` imposes no additional restriction. Consequently, fewer than the requested number may continue, or none may continue. For `"userDefined"`, implement any threshold condition in the supplied function. Efficacy and futility stopping rules apply separately from selection.

For example, the following multi-arm selection function drops one worst remaining arm at each enabled interim, while retaining at least one arm. Pass it as `selectArmsFunction = dropWorstArm` together with `typeOfSelection = "userDefined"`. If three arms are still available at the first interim, it retains two; at the next enabled interim it retains one, provided the trial continues.

```
dropWorstArm <- function(effectVector, selectedArms, stage) {
    keep <- selectedArms[seq_along(effectVector), stage] & !is.na(effectVector)
    if (sum(keep) > 1L) {
        remaining <- which(keep)
        worst <- remaining[which.min(effectVector[remaining])]
        keep[worst] <- FALSE
    }
    keep
}
```

Conditional power under piecewise hazards

Conditional power is calculated using a scalar hazard ratio estimated from the interim data or specified through `thetaH1`. The calculation uses a proportional-hazards working model, including when survival times are generated from piecewise hazards with interval-specific hazard ratios. The interval-specific hazards are not used to project future test-statistic drift in this calculation.

Accordingly, `conditionalPowerAchieved` reports conditional power under the scalar effect assumption for the selected number of events. It is not an empirical rejection probability and, when hazard ratios vary over time, need not equal the conditional rejection probability under the specified piecewise survival model. The default event-number recalculation uses the same working model; a custom `calcEventsFunction` can define a different recalculation rule.

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

```
## Not run:
# 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.
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 = 50,
    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 = 50,
    plannedEvents = c(75, 120))

y1$effectMatrix

y1$rejectAtLeastOne
y2$rejectAtLeastOne

y1$selectedArms
y2$selectedArms
```

```
# Non-inferiority with one active arm (H0: hazard ratio = 1.2)
simulationResultsNonInferiority <- getSimulationMultiArmSurvival(
    design = getDesignFixed(sided = 1), thetaH0 = 1.2,
    directionUpper = FALSE, activeArms = 1,
    typeOfShape = "userDefined", effectMatrix = matrix(1),
    piControl = 0.3, plannedEvents = 80,
    allocationRatioPlanned = 1, maxNumberOfSubjects = 400,
    accrualTime = c(0, 20), accrualIntensity = 20,
    maxNumberOfIterations = 50, simulationType = "patientWise")

# Piecewise exponential event times for two active arms and two situations
piecewiseTime <- getPiecewiseSurvivalTimeMultiArm(
    piecewiseSurvivalTime = c(0, 6),
    lambdaControl = c(0.08, 0.12),
    hazardRatio = list(
        favorable = rbind(lowDose = c(0.9, 0.8), highDose = c(0.8, 0.6)),
        conservative = rbind(lowDose = c(1, 0.9), highDose = c(0.9, 0.75))
    )
)
simulationResultsPiecewise <- getSimulationMultiArmSurvival(
    design = getDesignFixed(), simulationType = "patientWise",
    piecewiseSurvivalTime = piecewiseTime, directionUpper = FALSE,
    plannedEvents = 80, maxNumberOfSubjects = 300,
    accrualTime = c(0, 12), accrualIntensity = 25,
    maxNumberOfIterations = 50)

## End(Not run)
```

`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 = NA_real_,
  pi2 = NA_real_,
  plannedSubjects = NA_real_,
  directionUpper = NA,
  allocationRatioPlanned = NA_real_,
  minNumberOfSubjectsPerStage = NA_real_,
```

```
  maxNumberOfSubjectsPerStage = NA_real_,
  conditionalPower = NA_real_,
  pi1H1 = NA_real_,
  pi2H1 = NA_real_,
  maxNumberOfIterations = NA_integer_,
  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 general estimates, the normal case, and the binary case, 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 • *general estimates*: a value on the scale of the supplied estimates can be specified. • *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. • *count data*: a bound for testing H0: lambda1 / lambda2 = 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 numeric 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 Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.

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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.

minNumberOfSubjectsPerStage

When performing a data driven sample size recalculation, the numeric 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 numeric 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 maxNumberOfSubjectsPe (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; piTreatmentsH1 and piControlH1 in multi-arm designs, piTreatmentH1 and piControlH1 in enrichment designs), 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. Must be a positive integer of length 1.

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 and specified minNumberOfSubjectsPerStage and maxNumberOfSubjectsPerSt (see details and examples).

showStatistics
: Logical. 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, conditionalCriticalValu 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

```
## Not run:
# 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)

## End(Not run)
```

`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.

The outputs are matrices with stage rows and scenario columns for two-arm trials. Multi-arm treatment outputs are arrays with dimensions stage, scenario, and active arm; shared control is a stage-by-scenario matrix. Enrichment outputs have dimensions stage, scenario, and population (subpopulations followed by the full population). A single active arm or population uses a stage-by-scenario matrix. Stages and scenarios are never dropped, including for a fixed design. Discontinued groups contribute through their discontinuation analysis and stopped trials through their stopping analysis, with no later contributions. Overlapping enrichment populations include their shared subjects.

Usage

```
getSimulationSurvival(
  design = NULL,
  ...,
  thetaH0 = 1,
  directionUpper = NA,
  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 = NA_integer_,
  maxNumberOfRawDatasetsPerStage = 0,
  longTimeSimulationAllowed = FALSE,
  seed = NA_real_,
```

```
  calcEventsFunction = 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.
thetaH0	The null hypothesis value, default is 0 for general estimates, the normal case, and the binary case, 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 • *general estimates*: a value on the scale of the supplied estimates can be specified. • *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. • *count data*: a bound for testing H0: lambda1 / lambda2 = 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	Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.
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 c(0.1, 0.2, 0.3) (sample size calculations).
pi2	A numeric value that represents the assumed event rate in the control group, default is 0.5.
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). Must be a positive numeric value or vector: without piecewiseSurvivalTime, a vector defines alternative hazard rates to be considered; with piecewiseSurvivalTime, it contains one hazard rate per time interval. getEventProbabilities() uses only the first value of a non-piecewise vector.
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). Must be a positive numeric of length 1, or a vector with one hazard rate per time interval if piecewiseSurvivalTime is specified.

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. Must be a positive numeric of length 1.

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. Must be a positive numeric value or vector; `getEventProbabilities()` accepts a single value only.

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(q = time, shape = kappa, scale = 1 / lambda)` of the `stats` package.
For example, `getPiecewiseExponentialDistribution(time = 130, piecewiseLambda = 0.01, kappa = 4.2)` and `pweibull(q = 130, shape = 4.2, scale = 1 / 0.01)` provide the same 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 numeric 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`. For multi-arm simulations, this can be a vector of length equal to the number of active arms, specifying different drop-out rates for each active arm.

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 are specified, this will be calculated. Must be a positive integer of length 1.

plannedEvents `plannedEvents` is a numeric 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 numeric 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 numeric 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 `maxNumberOfSubjectsPe` (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; `piTreatmentsH1` and `piControlH1` in multi-arm designs, `piTreatmentH1` and `piControlH1` in enrichment designs), 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. Must be a numeric of length 1.

`maxNumberOfIterations`
: The number of simulation iterations, default is `1000`. Must be a positive integer of length 1.

`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.

`calcEventsFunction`
: Optionally, a function can be entered that defines the way of performing the sample size recalculation. By default, event number recalculation is performed with conditional power and specified `minNumberOfEventsPerStage` and `maxNumberOfEventsPerStage` (see details and examples).

`showStatistics`
: Logical. 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 the expected number of subjects.

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 variables stage, conditionalPower, thetaH0, plannedEvents, singleEventsPerStage, minNumberOfEventsPerStage, maxNumberOfEventsPerStage, allocationRatioPlanned, conditionalCriticalValue, The function has to contain the three-dots argument '...' (see examples).

The corresponding dropoutObservationTime1 and dropoutObservationTime2 fields retain the mean stage analysis time over exactly the same contributing replications, with the same dimensions as the proportions. They are omitted from print and summary output but can be accessed directly as result$dropoutObservationTime1 and result$dropoutObservationTime2. In multi-arm trials, the first field refers to the active treatment arms and the second to shared control. In enrichment trials, both fields contain population-specific results for their respective treatment arms. See SimulationResultsSurvival, SimulationResultsMultiArmSurvival, and SimulationResultsEnrichmentSurvival for the result fields and their dimensions.

These averages can differ from the trial-level analysisTime, because arm or population selection and the requirement for a nonempty enrolled-subject denominator can change which replications contribute. Within a replication, all arms or populations contributing at a given stage share its analysis time; differences between the reported averages arise from the contributing sets of replications, rather than different analysis schedules within that replication. When both input dropout rates are zero, the proportions are also omitted from print and summary output while remaining in the result object.

These outputs require patient histories and are unavailable for simulationType = "testStatisticBased". Existing recruitment and allocation mechanisms are retained. The input dropout probability at dropoutTime describes the dropout distribution without competing events; it is not the simulated observed proportion. Analytic expected proportions condition on deterministic planned accrual at each planned analysis time, whereas simulation includes random analysis times, early stopping and selection. Compare them only with matching stage, horizon, enrollment and allocation assumptions.

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. `singleEventsPerStage`: 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.

What `maxNumberOfRawDatasetsPerStage` **does**

When `maxNumberOfRawDatasetsPerStage` = `0` (the default), simulations run as usual but *no* patient-level ("raw") data are kept - only summary results.

If you set `maxNumberOfRawDatasetsPerStage` > `0`, rpact will **save up to that many full patient-level datasets *per stage*** (i.e., per interim/final look). Each saved dataset corresponds to one simulated iteration and contains all subject-wise records accrued up to the stage at which that iteration stopped. You can later retrieve these datasets with `getRawData()`.

Why "max" and not `numberOfRawDatasetsPerStage`**?**

The value is an *upper bound per stage*, not a fixed count. The actual number of datasets stored can be smaller because:

- Some iterations stop early (e.g., at stage 1), so later stages receive fewer datasets.
- The simulation might finish before reaching the cap due to other stopping or iteration limits.

As a result, the number specified is the **maximum possible** datasets saved *per stage*, not the exact number.

Conditional power under piecewise hazards

Conditional power is calculated using a scalar hazard ratio estimated from the interim data or specified through `thetaH1`. The calculation uses a proportional-hazards working model, including when survival times are generated from piecewise hazards with interval-specific hazard ratios. The interval-specific hazards are not used to project future test-statistic drift in this calculation.

Accordingly, `conditionalPowerAchieved` reports conditional power under the scalar effect assumption for the selected number of events. It is not an empirical rejection probability and, when hazard ratios vary over time, need not equal the conditional rejection probability under the specified piecewise survival model. The default event-number recalculation uses the same working model; a custom `calcEventsFunction` can define a different recalculation rule.

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

```
## Not run:
# 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
designGS <- getDesignGroupSequential(kMax = 2)
getSimulationSurvival(
    design = designGS,
    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
designGS2 <- getDesignGroupSequential(informationRates = c(0.4, 0.7, 1))
getSimulationSurvival(
    design = designGS2,
    pi1 = 0.2, pi2 = 0.3, eventTime = 24,
    plannedEvents = round(designGS2$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 = designGS, 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 = designGS,
    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 = designGS,
    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 = designGS,
    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 = designGS,
    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 = designGS,
    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 = designGS,
    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.
designIN <- getDesignInverseNormal(informationRates = c(0.4, 0.7, 1))

resultsWithSSR1 <- getSimulationSurvival(
    design = designIN,
    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 = designIN,
    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 = designIN,
    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
resultsWithSSRGS <- getSimulationSurvival(
    design = designGS2,
    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 reproducable results
identical(
    getSimulationSurvival(
        plannedEvents = 40, maxNumberOfSubjects = 200,
        seed = 99
    )$analysisTime,
    getSimulationSurvival(
        plannedEvents = 40, maxNumberOfSubjects = 200,
        seed = 99
    )$analysisTime
)

# Perform recalculation of number of events based on conditional power as above.
# The number of events is recalculated only in the first interim, the recalculated number
# is also used for the final stage. Here, we use the user defind calcEventsFunction as
# follows (note that the last stage value in minNumberOfEventsPerStage and maxNumberOfEventsPerStage
# has no effect):
myCalcEventsFunction <- function(...,
        stage, conditionalPower, estimatedTheta,
        plannedEvents, eventsOverStages,
        minNumberOfEventsPerStage, maxNumberOfEventsPerStage,
        conditionalCriticalValue) {
    theta <- max(1 + 1e-12, estimatedTheta)
    if (stage == 2) {
        requiredStageEvents <-
          max(0, conditionalCriticalValue + qnorm(conditionalPower))^2 * 4 / log(theta)^2
        requiredOverallStageEvents <- min(
            max(minNumberOfEventsPerStage[stage], requiredStageEvents),
            maxNumberOfEventsPerStage[stage]
        ) + eventsOverStages[stage - 1]
    } else {
       requiredOverallStageEvents <- 2 * eventsOverStages[stage - 1] - eventsOverStages[1]
    }
```

```
    return(requiredOverallStageEvents)
}
resultsWithSSR <- getSimulationSurvival(
    design = designIN,
    hazardRatio = seq(1, 2.6, 0.5),
    pi2 = 0.3,
    conditionalPower = 0.8,
    plannedEvents = c(58, 102, 146),
    minNumberOfEventsPerStage = c(NA, 44, 4),
    maxNumberOfEventsPerStage = 4 * c(NA, 44, 4),
    maxNumberOfSubjects = 800,
    calcEventsFunction = myCalcEventsFunction,
    seed = 1234,
    maxNumberOfIterations = 50
)

## End(Not run)
```

`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_,
  directionUpper = NA
)
```

Arguments

`design`	The trial design.
`dataInput`	The summary data used for calculating the test results. This is an element of `DatasetGeneral`, `DatasetMeans`, `DatasetRates`, or `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 general estimates, the normal case, and the binary case, 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

- *general estimates*: a value on the scale of the supplied estimates can be specified.

- *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.

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.

directionUpper Logical. Specifies the direction of the alternative, only applicable for one-sided testing; TRUE means that larger values of the test statistics yield smaller p-values. The default is TRUE for means, rates, and count data, and FALSE for survival data (power calculations, simulations, and analyses). In sample size calculations, the direction is derived from the specified alternative; a specified directionUpper must agree with it.

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(), getClosedConditionalD
getConditionalPower(), getConditionalRejectionProbabilities(), getFinalConfidenceInterval(),
getFinalPValue(), getRepeatedConfidenceIntervals(), getRepeatedPValues(), getTestActions()

Examples

```
## Not run:
design <- getDesignInverseNormal()
dataRates <- getDataset(
    n1      = c(10, 10),
    n2      = c(20, 20),
    events1 = c( 8, 10),
    events2 = c(10, 16))
getStageResults(design, dataRates)

## End(Not run)
```

---

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(), getClosedConditionalD
getConditionalPower(), getConditionalRejectionProbabilities(), getFinalConfidenceInterval(),
getFinalPValue(), getRepeatedConfidenceIntervals(), getRepeatedPValues(), getStageResults()

Examples

```
## Not run:
design <- getDesignInverseNormal(kMax = 2)
data <- getDataset(
    n      = c( 20,  30),
    means  = c( 50,  51),
    stDevs = c(130, 140)
)
getTestActions(getStageResults(design, dataInput = data))

## End(Not run)
```

---

knit_print.SummaryFactory

Print Summary Factory in Markdown Code Chunks

Description

The function knit_print.SummaryFactory is the default printing function for rpact summary objects in knitr. The chunk option render uses this function by default. To fall back to the normal printing behavior set the chunk option render = normal_print. For more information see knit_print.

Usage

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

Arguments

x	A SummaryFactory.
...	Other arguments (see knit_print).

Details

Generic function to print a summary object in Markdown.

Markdown options

Use `options("rpact.print.heading.base.number" = NUMBER)` (where NUMBER is an integer value >= -2) to specify the heading level.

NUMBER = 1 results in the heading prefix #, NUMBER = 2 results in ##, ...

The default is `options("rpact.print.heading.base.number" = -2)`, i.e., the top headings will be written italic but are not explicit defined as header. `options("rpact.print.heading.base.number" = -1)` means that all headings will be written bold but are not explicit defined as header.

Furthermore the following options can be set globally:

- `rpact.auto.markdown.all`: if TRUE, all output types will be rendered in Markdown format automatically.
- `rpact.auto.markdown.print`: if TRUE, all print outputs will be rendered in Markdown format automatically.
- `rpact.auto.markdown.summary`: if TRUE, all summary outputs will be rendered in Markdown format automatically.
- `rpact.auto.markdown.plot`: if TRUE, all plot outputs will be rendered in Markdown format automatically.

Example: `options("rpact.auto.markdown.plot" = FALSE)` disables the automatic knitting of plots inside Markdown documents.

---

knit_print.TrialComparisonTable

Render a Trial Comparison Table in knitr

Description

Renders data frames prepared by `as.data.frame.TrialComparison()` with `output = "auto"`, `"html"`, or `"pdf"`. HTML tables scroll horizontally. LaTeX tables use longtable and split wide results into column blocks, repeating the identifier columns in each block.

Usage

```
## S3 method for class 'TrialComparisonTable'
knit_print(x, ...)
```

Arguments

x	A `TrialComparisonTable` returned by `as.data.frame.TrialComparison()`.
...	Further arguments passed to the ordinary knitr data-frame renderer when the target is neither HTML nor LaTeX.

Details

PDF mode emits a LaTeX fragment for the surrounding document. It requires the suggested rmarkdown package and declares the longtable dependency via knitr metadata. No PDF file is created directly by this method.

Value

A `knitr::asis_output` object for HTML or LaTeX. Other targets use the ordinary knitr data-frame representation. Empty tables emit no output.

`mvnprd` *Original Algorithm AS 251: Normal Distribution*

Description

Calculates the Multivariate Normal Distribution with Product Correlation Structure published by Charles Dunnett, Algorithm AS 251.1 Appl.Statist. (1989), Vol.38, No.3, doi:10.2307/2347754.

Usage

```
mvnprd(..., A, B, BPD, EPS = 1e-06, INF, IERC = 1, HINC = 0)
```

Arguments

`...`	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
`A`	Upper limits of integration. Array of N dimensions
`B`	Lower limits of integration. Array of N dimensions
`BPD`	Values defining correlation structure. Array of N dimensions
`EPS`	desired accuracy. Defaults to 1e-06
`INF`	Determines where integration is done to infinity. Array of N dimensions. Valid values for INF(I): 0 = c(B(I), Inf), 1 = c(-Inf, A(I)), 2 = c(B(I), A(I))
`IERC`	error control. If set to 1, strict error control based on fourth derivative is used. If set to zero, error control based on halving intervals is used
`HINC`	Interval width for Simpson's rule. Value of zero caused a default .24 to be used

Details

This is a wrapper function for the original Fortran 77 code. For a multivariate normal vector with correlation structure defined by RHO(I,J) = BPD(I) * BPD(J), computes the probability that the vector falls in a rectangle in n-space with error less than eps.

mvstud *Original Algorithm AS 251: Student T Distribution*

Description

Calculates the Multivariate Normal Distribution with Product Correlation Structure published by Charles Dunnett, Algorithm AS 251.1 Appl.Statist. (1989), Vol.38, No.3, doi:10.2307/2347754.

Usage

```
mvstud(..., NDF, A, B, BPD, D, EPS = 1e-06, INF, IERC = 1, HINC = 0)
```

Arguments

...	Ensures that all arguments (starting from the "...") are to be named and that a warning will be displayed if unknown arguments are passed.
NDF	Degrees of Freedom. Use 0 for infinite D.F.
A	Upper limits of integration. Array of N dimensions
B	Lower limits of integration. Array of N dimensions
BPD	Values defining correlation structure. Array of N dimensions
D	Non-Centrality Vector
EPS	desired accuracy. Defaults to 1e-06
INF	Determines where integration is done to infinity. Array of N dimensions. Valid values for INF(I): 0 = c(B(I), Inf), 1 = c(-Inf, A(I)), 2 = c(B(I), A(I))
IERC	error control. If set to 1, strict error control based on fourth derivative is used. If set to zero, error control based on halving intervals is used
HINC	Interval width for Simpson's rule. Value of zero caused a default .24 to be used

Details

This is a wrapper function for the original Fortran 77 code. For a multivariate normal vector with correlation structure defined by RHO(I,J) = BPD(I) * BPD(J), computes the probability that the vector falls in a rectangle in n-space with error less than eps.

Examples

```
## Not run:
N <- 3
RHO <- 0.5
B <- rep(-5.0, length = N)
A <- rep(5.0, length = N)
INF <- rep(2, length = N)
BPD <- rep(sqrt(RHO), length = N)
D <- rep(0.0, length = N)
result <- mvstud(NDF = 0, A = A, B = B, BPD = BPD, INF = INF, D = D)
result

## End(Not run)
```

obtain *Extract a single parameter*

Description

Fetch a parameter from a parameter set.

Usage

```
obtain(x, ..., output)

## S3 method for class 'ParameterSet'
obtain(x, ..., output = c("named", "labeled", "value", "list"))

fetch(x, ..., output)

## S3 method for class 'ParameterSet'
fetch(x, ..., output = c("named", "labeled", "value", "list"))
```

Arguments

x	The ParameterSet object to fetch from.
...	One or more variables specified as: • a literal variable name • a positive integer, giving the position counting from the left • a negative integer, giving the position counting from the right. The default returns the last parameter. This argument is taken by expression and supports quasi-quotation (you can unquote column names and column locations).
output	A character defining the output type as follows: • "named" (default) returns the named value if the value is a single value, the value inside a named list otherwise • "value" returns only the value itself • "list" returns the value inside a named list

See Also

fetch.TrialComparison() for selecting contained objects from a comparison.

Examples

```
## Not run:
getDesignInverseNormal() |> fetch(kMax)
getDesignInverseNormal() |> fetch(kMax, output = "list")

## End(Not run)
```

`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 = "auto",
  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 subpopulation.

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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.

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 legend position. The default, "auto", measures the legend and chooses among six inside positions by minimizing overlap with points, line segments, reference lines, bars, boxes, and error bars. Long labels are wrapped and entries arranged vertically in automatic mode. Oversized legends and faceted plots use the outside right position. This heuristic uses the current graphics device (seven inches square when no device is open); resizing may require recreating the plot or setting a position explicitly. Manual positions are:

- -1 or "none": hide the legend
- "left", "right", "bottom", "top": outside the plot
- 0: default outside position with a square aspect ratio
- 1, 2, 3: inside left, at the top, center, or bottom
- 4, 5, 6: inside right, at the top, center, or bottom
- NA: retain the legacy plot-specific positioning rules

showSource — Logical. 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

```
## Not run:
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))

## End(Not run)
```

---

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,
  legendPosition = "auto"
)
```

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	Logical. 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().
legendPosition	The legend position. The default, "auto", measures the legend and chooses among six inside positions by minimizing overlap with points, line segments, reference lines, bars, boxes, and error bars. Long labels are wrapped and entries arranged vertically in automatic mode. Oversized legends and faceted plots use the outside right position. This heuristic uses the current graphics device (seven inches square when no device is open); resizing may require recreating the plot or setting a position explicitly. Manual positions are: • -1 or "none": hide the legend • "left", "right", "bottom", "top": outside the plot • 0: default outside position with a square aspect ratio • 1, 2, 3: inside left, at the top, center, or bottom • 4, 5, 6: inside right, at the top, center, or bottom • NA: retain the legacy plot-specific positioning rules

Details

Generic function to plot all kinds of datasets.

Value

Returns a ggplot2 object.

Examples

```
## Not run:
# 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")

## End(Not run)
```

---

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 = "auto",
  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. For simulating means and rates for a two treatment groups design, it can be a vector of length kMax, the number of stages. It can be a vector of length kMax, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally allocationRatioPlanned is treated as a vector of length kMax, not a scalar.
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	Logical. If TRUE, additional points will be plotted.
legendPosition	The legend position. The default, "auto", measures the legend and chooses among six inside positions by minimizing overlap with points, line segments, reference lines, bars, boxes, and error bars. Long labels are wrapped and entries arranged vertically in automatic mode. Oversized legends and faceted plots use the outside right position. This heuristic uses the current graphics device (seven inches square when no device is open); resizing may require recreating the plot or setting a position explicitly. Manual positions are: • -1 or "none": hide the legend • "left", "right", "bottom", "top": outside the plot • 0: default outside position with a square aspect ratio • 1, 2, 3: inside left, at the top, center, or bottom • 4, 5, 6: inside right, at the top, center, or bottom • NA: retain the legacy plot-specific positioning rules
showSource	Logical. 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 an event probabilities object.

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 = "auto",
  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
: Logical. If TRUE, additional points will be plotted.

legendPosition
: The legend position. The default, "auto", measures the legend and chooses among six inside positions by minimizing overlap with points, line segments, reference lines, bars, boxes, and error bars. Long labels are wrapped and entries arranged vertically in automatic mode. Oversized legends and faceted plots use the outside right position. This heuristic uses the current graphics device (seven inches square when no device is open); resizing may require recreating the plot or setting a position explicitly. Manual positions are:

  - -1 or "none": hide the legend
  - "left", "right", "bottom", "top": outside the plot
  - 0: default outside position with a square aspect ratio
  - 1, 2, 3: inside left, at the top, center, or bottom
  - 4, 5, 6: inside right, at the top, center, or bottom
  - NA: retain the legacy plot-specific positioning rules

showSource
: Logical. 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 "number of subjects" object.

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 = "auto",
  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 legend position. The default, `"auto"`, measures the legend and chooses among six inside positions by minimizing overlap with points, line segments, reference lines, bars, boxes, and error bars. Long labels are wrapped and entries arranged vertically in automatic mode. Oversized legends and faceted plots use the outside right position. This heuristic uses the current graphics device (seven inches square when no device is open); resizing may require recreating the plot or setting a position explicitly. Manual positions are: • `-1` or `"none"`: hide the legend • `"left"`, `"right"`, `"bottom"`, `"top"`: outside the plot • `0`: default outside position with a square aspect ratio • `1`, `2`, `3`: inside left, at the top, center, or bottom

- 4, 5, 6: inside right, at the top, center, or bottom
- NA: retain the legacy plot-specific positioning rules

showSource — Logical. 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 = NA_integer_,
  palette = "Set1",
  theta = seq(-1, 1, 0.01),
  plotPointsEnabled = NA,
  legendPosition = "auto",
  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	Logical. If TRUE, additional points will be plotted.
legendPosition	The legend position. The default, "auto", measures the legend and chooses among six inside positions by minimizing overlap with points, line segments, reference lines, bars, boxes, and error bars. Long labels are wrapped and entries arranged vertically in automatic mode. Oversized legends and faceted plots use the outside right position. This heuristic uses the current graphics device (seven inches square when no device is open); resizing may require recreating the plot or setting a position explicitly. Manual positions are: • -1 or "none": hide the legend • "left", "right", "bottom", "top": outside the plot

- 0: default outside position with a square aspect ratio
- 1, 2, 3: inside left, at the top, center, or bottom
- 4, 5, 6: inside right, at the top, center, or bottom
- NA: retain the legacy plot-specific positioning rules

showSource — Logical. 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

For piecewise multi-arm and enrichment survival simulations, scenario plots use situation positions on the x-axis, labeled with the specification's situation names, including simulations with one arm or one subgroup. First-interval hazard ratios are not used as scenario coordinates.

Generic function to plot all kinds of simulation results.

Value

Returns a ggplot2 object.

Examples

```
## Not run:
results <- getSimulationMeans(
    alternative = 0:4, stDev = 5,
    plannedSubjects = 40, maxNumberOfIterations = 1000
)
plot(results, type = 5)

## End(Not run)
```

`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 = "auto",
  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 subpopulation.

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. For simulating means and rates for a two treatment groups design, it can be a vector of length `kMax`, the number of stages. It can be a vector of length `kMax`, too, for multi-arm and enrichment designs. In these cases, a change of allocating subjects to treatment groups over the stages can be assessed. Note that internally `allocationRatioPlanned` is treated as a vector of length `kMax`, not a scalar.

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 legend position. The default, `"auto"`, measures the legend and chooses among six inside positions by minimizing overlap with points, line segments, reference lines, bars, boxes, and error bars. Long labels are wrapped and entries arranged vertically in automatic mode. Oversized legends and faceted plots use the outside right position. This heuristic uses the current graphics device (seven inches square when no device is open); resizing may require recreating the plot or setting a position explicitly. Manual positions are:

  - `-1` or `"none"`: hide the legend
  - `"left"`, `"right"`, `"bottom"`, `"top"`: outside the plot
  - `0`: default outside position with a square aspect ratio
  - `1`, `2`, `3`: inside left, at the top, center, or bottom
  - `4`, `5`, `6`: inside right, at the top, center, or bottom
  - `NA`: retain the legacy plot-specific positioning rules

showSource
: Logical. 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

```
## Not run:
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))

## End(Not run)
```

---

plot.SummaryFactory *Summary Factory Plotting*

Description

Plots a summary factory.

Usage

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

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.
showSummary	Show the summary before creating the plot output, default is FALSE.

Details

Generic function to plot all kinds of summary factories.

Value

Returns a ggplot2 object.

`plot.TrialComparison` *Plot a Trial Comparison*

Description

Plots selected variables from a trial comparison.

The summary method plots the objects contained in the comparison.

Usage

```
## S3 method for class 'TrialComparison'
plot(
  x,
  xAxis = NULL,
  yAxis = NULL,
  ...,
  data = NULL,
  category = NULL,
  group = NULL,
  main = NULL,
  xlab = NA_character_,
  ylab = NA_character_
)

## S3 method for class 'TrialComparisonSummary'
plot(
  x,
  xAxis = NULL,
  yAxis = NULL,
  ...,
  showSummary = FALSE,
  category = NULL,
  group = NULL,
  data = NULL
)
```

Arguments

x	A TrialComparison or its summary.
xAxis, yAxis	Optional axis column names or captions. Omitted axes are selected only when there is exactly one candidate: inputs/design settings are preferred for x, results for y, then other varying numeric columns. Identifier columns are excluded; stages remain eligible axes. yAxis may also explicitly select multiple numeric parameters. Their axis sides can be controlled through yAxisSide passed to plot.TrialComparison().
...	Arguments passed to plot.TrialComparison(), including categories, grouping, titles and line/point settings. Explicit axis selections prevail.

data	Optional comparison data frame with additional annotation columns. Rows must follow the order of as.data.frame(x). Without custom data, summary plots retain the summary's column-name harmonization setting.
category	Optional column names defining categories. Multiple columns define joint categories. NULL selects automatically; NA_character_ or character(0) disables categories.
group	Optional column name defining an additional grouping.
main	Plot title. When omitted, generated from the original objects. NULL or NA hides the title.
xlab, ylab	Axis labels. NA uses automatic parameter captions; NULL hides the label. ylab overrides the primary y-axis label, including a single right-positioned axis. A secondary y axis retains its automatic caption.
showSummary	If TRUE, also print the summary before creating its plot.

Details

xAxis, yAxis, category, and group accept technical column names, captions from the parameter dictionary, and object-specific captions as in fetch(). For example, "Alternative" and "Alternatives" both select alternative. Names may also be unquoted, including inside c(...). Defined variables and other expressions are evaluated in the calling environment. Exact column names take precedence over captions. Other names must resolve unambiguously to a column present in the comparison export; harmonized names are taken into account. With fixed plans alone, "Number of subjects" can select nFixed. Unknown names, ambiguous captions and duplicate selections (including two aliases for one column) raise selection errors.

Value

A ggplot object. Ambiguous selections raise an error listing candidate descriptions and a concrete example.

Examples

```
comparison <- getSampleSizeMeans(alternative = 0.22, stDev = 0.5) |>
    compare(getSampleSizeMeans(alternative = 0.44, stDev = 1)) |>
    compare(getSampleSizeMeans(alternative = 0.59, stDev = 1))
plot(comparison,
    xAxis = "Alternative",
    yAxis = c("Number of subjects", "Critical values (treatment effect scale)"),
    category = "Standard deviation"
)
plot(comparison, xAxis = alternative, yAxis = numberOfSubjects, group = stDev)
```

---

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

## S3 method for class 'TrialDesignCharacteristics'
plot(x, y, ..., type = 1L, grid = 1)
```

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. Must be a positive integer of length 1.

plotPointsEnabled — Logical. If TRUE, additional points will be plotted.

legendPosition — The legend position. The default, "auto", measures the legend and chooses among six inside positions by minimizing overlap with points, line segments, reference lines, bars, boxes, and error bars. Long labels are wrapped and entries arranged vertically in automatic mode. Oversized legends and faceted plots use the outside right position. This heuristic uses the current graphics device (seven inches square when no device is open); resizing may require recreating the plot or setting a position explicitly. Manual positions are:

- -1 or "none": hide the legend
- "left", "right", "bottom", "top": outside the plot
- 0: default outside position with a square aspect ratio
- 1, 2, 3: inside left, at the top, center, or bottom
- 4, 5, 6: inside right, at the top, center, or bottom
- NA: retain the legacy plot-specific positioning rules

showSource — Logical. 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() to compare different designs or design parameters visual.

Examples

```
## Not run:
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
}

## End(Not run)
```

---

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 = NA_integer_,
  palette = "Set1",
  theta = NA_real_,
  plotPointsEnabled = NA,
  legendPosition = "auto",
  showSource = FALSE,
  grid = 1,
  plotSettings = NULL
)
```

Arguments

x	The trial design plan, obtained from getSampleSizeMeans(), getSampleSizeRates(), getSampleSizeSurvival(),

getSampleSizeCounts(),
getPowerMeans(),
getPowerRates() or
getPowerSurvival() or
getPowerCounts().

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 — Logical. If TRUE, additional points will be plotted.

legendPosition — The legend position. The default, "auto", measures the legend and chooses among six inside positions by minimizing overlap with points, line segments, reference lines, bars, boxes, and error bars. Long labels are wrapped and entries arranged vertically in automatic mode. Oversized legends and faceted plots use the outside right position. This heuristic uses the current graphics device (seven inches square when no device is open); resizing may require recreating the plot or setting a position explicitly. Manual positions are:

- -1 or "none": hide the legend
- "left", "right", "bottom", "top": outside the plot
- 0: default outside position with a square aspect ratio
- 1, 2, 3: inside left, at the top, center, or bottom
- 4, 5, 6: inside right, at the top, center, or bottom
- NA: retain the legacy plot-specific positioning rules

showSource — Logical. 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

```
## Not run:
if (require(ggplot2)) plot(getSampleSizeMeans())

## End(Not run)
```

---

plot.TrialDesignSet *Trial Design Set Plotting*

Description

Plots a trial design set.

Usage

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

```
  type = NA_integer_,
  main = NA_character_,
  xlab = NA_character_,
  ylab = NA_character_,
  palette = "Set1",
  theta = seq(-1, 1, 0.02),
  nMax = NA_integer_,
  plotPointsEnabled = NA,
  legendPosition = "auto",
  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. Must be a positive integer of length 1.
plotPointsEnabled	Logical. If TRUE, additional points will be plotted.
legendPosition	The legend position. The default, "auto", measures the legend and chooses among six inside positions by minimizing overlap with points, line segments, reference lines, bars, boxes, and error bars. Long labels are wrapped and entries arranged vertically in automatic mode. Oversized legends and faceted plots use the outside right position. This heuristic uses the current graphics device (seven inches square when no device is open); resizing may require recreating the plot or setting a position explicitly. Manual positions are: • -1 or "none": hide the legend

- "left", "right", "bottom", "top": outside the plot
- 0: default outside position with a square aspect ratio
- 1, 2, 3: inside left, at the top, center, or bottom
- 4, 5, 6: inside right, at the top, center, or bottom
- NA: retain the legacy plot-specific positioning rules

showSource — Logical. 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

```
## Not run:
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)

## End(Not run)
```

plot.TrialDesignSummaries *Plot Trial Design Summaries*

Description

Generic function to plot a `TrialDesignSummaries` object.

Usage

```
## S3 method for class 'TrialDesignSummaries'
plot(x, ..., type = 1L, grid = 1)
```

Arguments

`x`	a `TrialDesignSummaries` object to plot.
`...`	further arguments passed to or from other methods.
`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
`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'.

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

Returns a list if option is either capnum or numcap or returns a vector that is of character type for option=caption or of numeric type for option=numeric.

Examples

```
## Not run:
design <- getDesignInverseNormal(kMax = 2)
getAvailablePlotTypes(design, "numeric")
plotTypes(design, "caption")
getAvailablePlotTypes(design, "numcap")
plotTypes(design, "capnum")

## End(Not run)
```

print.SummaryFactory *Summary Factory Printing*

Description

Prints the result object stored inside a summary factory.

Usage

```
## S3 method for class 'SummaryFactory'
print(x, ..., markdown = NA, sep = NA_character_)
```

Arguments

x	The summary factory object.
...	Optional plot arguments. At the moment xlim and ylim are implemented for changing x or y axis limits without dropping data observations.
markdown	If TRUE, the object x will be printed using markdown syntax; normal representation will be used otherwise (default is FALSE)
sep	The separator line between the summary and the print output, default is "\n\n-----\n\n".

Details

Generic function to print all kinds of summary factories.

print.TrialComparison *Print a Trial Comparison*

Description

Prints a compact overview of a trial comparison using descriptive parameter captions.

Usage

```
## S3 method for class 'TrialComparison'
print(
  x,
  ...,
  markdown = NA,
  digits = NA_integer_,
  maxTypes = 5L,
  maxMetrics = 6L
)
```

Arguments

x	A TrialComparison.
...	Further arguments. Currently unused.
markdown	Whether to return Markdown output. NA uses rpact settings.
digits	Display rounding only, as in getComparisonParameterDescriptions().
maxTypes, maxMetrics	Maximum numbers of object types and selected metrics to display. These limits do not affect as.data.frame.TrialComparison().

Value

The object invisibly, or knitr Markdown output when requested.

---

print.TrialDesignCharacteristics
Trial Design Characteristics Printing

Description

Prints the design characteristics object.

Usage

```
## S3 method for class 'TrialDesignCharacteristics'
print(x, ..., markdown = NA, showDesign = TRUE)
```

Arguments

x	The trial design characteristics object.
...	Optional plot arguments. At the moment xlim and ylim are implemented for changing x or y axis limits without dropping data observations.
markdown	If TRUE, the object x will be printed using markdown syntax; normal representation will be used otherwise (default is FALSE)
showDesign	Show the design print output above the design characteristics, default is TRUE.

Details

Generic function to print all kinds of design characteristics.

print.TrialDesignSummaries *Print Trial Design Summaries*

Description

Generic function to print a `TrialDesignSummaries` object.

Usage

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

Arguments

`x`	a `TrialDesignSummaries` object to print.
`...`	further arguments passed to or from other methods.

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(),
  tolerance = 1e-07,
  pipeOperator = c("auto", "none", "magrittr", "R"),
  output = c("vector", "cat", "test", "markdown", "internal"),
  explicitPrint = FALSE,
  showVersus = FALSE
)

getObjectRCode(
  obj,
  ...,
  leadingArguments = NULL,
  includeDefaultParameters = FALSE,
```

```
  stringWrapParagraphWidth = 90,
  prefix = "",
  postfix = "",
  stringWrapPrefix = "",
  newArgumentValues = list(),
  tolerance = 1e-07,
  pipeOperator = c("auto", "none", "magrittr", "R"),
  output = c("vector", "cat", "test", "markdown", "internal"),
  explicitPrint = FALSE,
  showVersus = FALSE
)
```

Arguments

obj	The result object, including a TrialComparison or its summary. Comparison reconstruction inlines each object's preconditions. Argument overrides are unsupported for comparisons, and their code is not word-wrapped so object names and callback bodies remain intact.
...	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.
pipeOperator	The pipe operator to use in the R code, default is "none".
output	The output format, default is a character "vector".
explicitPrint	Show an explicit print command, default is FALSE.
showVersus	If TRUE, reconstruct a TrialComparisonSummary as a chain using %vs%. Default is FALSE. A summary() call is appended only for non-default display settings or when includeDefaultParameters = TRUE. If omitted, use option rpact.r.code.show.versus or environment variable RPACT_R_CODE_SHOW_VERSUS (the environment variable takes precedence). Explicit arguments override both settings. Other object classes are unaffected. includeDefaultParameters also controls the contained objects.

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 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 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

```
## Not run:
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
}

## End(Not run)
```

`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

```
## Not run:
dataFile <- system.file("extdata", "datasets_rates.csv", package = "rpact")
if (dataFile != "") {
    datasets <- readDatasets(dataFile)
    datasets
}

## End(Not run)
```

---

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

Description

rpact (R Package for Adaptive Clinical Trials) supports the design, simulation, and analysis of confirmatory clinical trials. It provides sample size and power calculations for standard fixed-sample and classical group sequential designs, with methods for continuous, binary, survival (time-to-event), and count data endpoints. Adaptive methods include multi-stage combination tests, sample size reassessment, multi-arm multi-stage (MAMS) designs, and population enrichment designs. The package implements a broad range of methods described by Wassmer and Brannath (2025).

Details

Planning and simulation functions are available for means, rates, survival, and count data. Survival planning supports staggered recruitment and piecewise exponentially distributed survival times. Multi-arm and population enrichment simulation and analysis methods are available for means, rates, and survival endpoints. General estimates and standard errors can also be analyzed through `getDataset()` and `getAnalysisResults()`.

Classical group sequential designs specify rules for interim and final analyses. Alpha-spending approaches allow boundary recalculation for observed information, including under- and over-running. Adaptations based on interim treatment effects require methods that preserve Type I error control, such as combination testing or the conditional rejection probability (CRP) principle. The package provides inverse normal and Fisher combination tests and conditional Dunnett methods. Simulations can assess the operating characteristics of sample size or event-number reassessment strategies.

Analysis methods provide p-values, conditional power, and confidence intervals where supported by the selected design and endpoint. In particular, final confidence intervals and median unbiased estimates are not available for Fisher combination tests or multi-arm and enrichment analyses. See `getAnalysisResults()` for further conditions and limitations. `getFutilityBounds()` converts futility boundaries between different scales, including conditional and predictive power scales.

rpact is free of charge and open source under the LGPL-3 license. Package information and tutorials are available at www.rpact.org; function documentation is available at docs.rpact.org.

RPACT GmbH develops Statistical Tools for Drug Development and offers software development, validation, technical support, consulting, and training. Formal validation documentation is available to customers and supporting members. It includes credentials for the full installation qualification test suite run by `testPackage()`. For services, see www.rpact.com. Browser-based tools are available at demo.rpact.cloud (RPACT Cloud) and connect.rpact.app (RPACT Connect).

rpact is developed by

- Gernot Wassmer (<gernot.wassmer@rpact.com>),
- Friedrich Pahlke (<friedrich.pahlke@rpact.com>), and
- Daniel Sabanés Bové (<daniel.sabanes_bove@rpact.com>).

Author(s)

Gernot Wassmer, Friedrich Pahlke

References

Wassmer, G., Brannath, W. (2025) Group Sequential and Confirmatory Adaptive Designs in Clinical Trials (Springer Series in Pharmaceutical Statistics; doi:10.1007/9783031896699)

See Also

Package website, Function reference, RPACT Cloud, RPACT Connect, RPACT GmbH, and Bug reports.

`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_,
  persist = TRUE
)
```

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_.
persist	A logical value indicating whether the output format settings should be saved persistently. Default is TRUE.

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").

Drop-out observation times use a separate output format with the same default precision as study duration. For example, use setOutputFormat("dropoutObservationTime1", digits = 4, nsmall = 4) to change it for both arms without changing other time formats.

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

```
## Not run:
# 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")

## End(Not run)
```

`summary.TrialComparison`

Summarize Differences Between Trial Planning Objects

Description

Summarizes differences between the object profiles in a trial comparison.

The `TrialComparisonSummary` method refreshes a summary from its underlying comparison, retaining stored display settings unless explicitly overridden. This also supports `%vs%` chains followed by `summary(...)`.

Usage

```
## S3 method for class 'TrialComparison'
summary(
  object,
  ...,
  digits = NA_integer_,
  maxParameters = NA_integer_,
  maxProfiles = NA_integer_,
  maxObjects = NA_integer_,
  maxValues = NA_integer_,
  output = c("all", "title", "overview", "body"),
  harmonizeColumnNames = TRUE,
  textMode = c("wrap", "truncate", "full"),
```

```
  width = NA_integer_,
  showAll = NA,
  showDataType = FALSE,
  maxTableColumns = NA_integer_
)

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

Arguments

object	A TrialComparison or an existing TrialComparisonSummary.
...	Further arguments. Currently unused.
digits	Display rounding only; exact values determine differences.
maxParameters	Maximum number of differing parameters shown, across all sections. Increase this to inspect further differences.
maxProfiles	Maximum number of distinct profiles shown per parameter.
maxObjects	Maximum object numbers/names listed per profile.
maxValues	Maximum profile values shown per preview.
output	One of "all", "title", "overview", or "body".
harmonizeColumnNames	If TRUE (default), compare the harmonized columns described in getComparison(). If FALSE, compare the original parameter names, as in as.data.frame(object, harmonizeColumnNames = FALSE).
textMode	Console text handling: "wrap" (default) wraps complete text, "truncate" shows compact previews with an explicit shortening notice, and "full" writes complete text without automatic wrapping. Markdown always retains complete text in its profile tables.
width	Console line width, at least 20 characters. Defaults to getOption("width", 80L). Long unbroken words can exceed this width.
showAll	If TRUE, lift all four max* limits and disable text truncation. Wrapping remains active when textMode = "wrap". Defaults to FALSE.
showDataType	If TRUE, include data type descriptions such as integer, continuous in parameter headings. Defaults to FALSE.
maxTableColumns	Maximum result/profile columns per Markdown table block (default 4). Coordinate columns are additional. Also applies with showAll = TRUE.

Details

Constant numeric vector cells compare equal to their scalar value regardless of vector length. Non-constant vectors, missing values, and their positions remain part of the exact comparison. The stored comparison data are unchanged. Profile coordinates follow the parameter's source and native dimensions. Repeated design settings on simulation category rows are normalized only when the remaining coordinates uniquely determine the exact values. Stage and group domains of dimensioned parameters remain meaningful when their values are equal. Uncertain or conflicting dependencies retain additional coordinates rather than averaging or selecting an arbitrary value. Both parameter-filter options of as.data.frame.TrialComparison() use these profiles; unfiltered data-frame rows remain unchanged for plotting. This value normalization also applies when

column-name harmonization is disabled. Plotting a summary retains its selected column-name mode unless custom data are supplied.

Parameter captions are resolved separately for each original object, including its design when a parameter comes from that design. `parameterInfo$paramName` retains the column names selected by `harmonizeColumnNames`; `paramCaption` contains the display label and `paramCaptions` contains captions in object order (NA for absent parameters). These captions are captured with the summary. Distinct object-specific captions are shown together and identified alongside the corresponding object labels; they do not change exact value-profile grouping. Unnamed entries are labelled `Simulation` when all objects are simulation results, `Design` when all objects are trial designs, and `Result` otherwise, followed by their collection position. Explicit names take precedence. Full type descriptions remain in the overview. The overview and these labels are captured with the summary. The legend explains exact profile comparison and mentions NA only when displayed profiles contain missing values. Titles distinguish designs, sample size calculations, power calculations, and power simulations. Mixed comparisons use a design plan comparison title; empty comparisons retain the general trial comparison title. Markdown output places coordinates in rows and displayed profiles in columns. Scalar profiles use one-row tables. At most `maxTableColumns` profile columns (default 4) appear per block, with coordinate columns repeated in each block. This layout limit also applies with `showAll = TRUE`. Long profile labels are listed above the tables with short references. Incompatible coordinate structures use separate tables. Missing and omitted coordinates use the same markers as console tables. Data limits apply independently of table layout.

Console output uses a single comparison line for numeric, character or logical scalar values when at most four objects are compared, all fit within the profile/object limits, captions agree and the complete line fits within `width`. Otherwise scalars retain the labelled profile display. Profiles with compatible coordinate columns are aligned by their exact coordinates and displayed side by side. Coordinates constant across the complete parameter are listed once, except redundant constant category types, categories and situation identifiers, which are omitted. Varying categories use an Arm or Comparison column caption where appropriate. A scalar missing profile is displayed as an `NA` column alongside the coordinates of the other profiles; non-missing scalars are not expanded. If the table exceeds `width`, values are listed vertically per coordinate. Profiles sharing at least one coordinate can use a joint table even if their remaining axes differ. A dash marks an inapplicable coordinate; values are not broadcast to categories absent from their source profile. Profiles without shared coordinates retain separate displays. `<absent>` denotes a coordinate absent from a profile, whereas `<omitted>` denotes a value outside that profile's `maxValues` preview. Neither marker represents a stored `NA`. Display choices do not change exact profile grouping.

A `NULL` display argument reads its setting through `.getEnvironmentVariable()`. Explicit arguments take precedence over environment variables, which take precedence over R options and defaults. The four default limits are 12 parameters, 3 profiles, 6 objects and 4 values. R options use the prefix `rpact.comparison.summary.` followed by the argument name: `maxParameters`, `maxProfiles`, `maxObjects`, `maxValues`, `maxTableColumns`, `textMode`, `width`, `showAll` or `showDataType`. The corresponding environment variables are `RPACT_COMPARISON_SUMMARY_MAX_PARAMETERS`, `RPACT_COMPARISON_SUM` `RPACT_COMPARISON_SUMMARY_MAX_OBJECTS`, `RPACT_COMPARISON_SUMMARY_MAX_VALUES`, `RPACT_COMPARISON_SUMMAR` `RPACT_COMPARISON_SUMMARY_WIDTH`, `RPACT_COMPARISON_SUMMARY_SHOW_ALL` and `RPACT_COMPARISON_SUMMARY_SHO` Settings are captured when the summary is created. For a complete display use `summary(x, showAll = TRUE)`; for a compact display use `textMode = "truncate"`.

Value

A `TrialComparisonSummary` inheriting from `SummaryFactory`. Its `parameterInfo` contains metadata, full exact profiles and object group IDs. The displayed summary is a snapshot; call `summary()` again after changing an underlying object. The `harmonizeColumnNames` attribute of `parameterInfo` records the selected naming mode for plotting and R-code reconstruction. Input roles take precedence when roles differ across the compared object types.

testPackage *Test and Validate the rpact Package Installation*

Description

This function ensures the correct installation of the rpact package by performing various tests. It supports a comprehensive validation process, essential for GxP compliance and other regulatory requirements.

Usage

```
testPackage(
  outDir = ".",
  ...,
  completeUnitTestSetEnabled = TRUE,
  connection = list(token = NULL, secret = NULL),
  testFileDirectory = NA_character_,
  downloadTestsOnly = FALSE,
  addWarningDetailsToReport = TRUE,
  reportType = c("compact", "detailed", "Rout"),
  testInstalledBasicPackages = TRUE,
  scope = c("basic", "devel", "both", "internet", "all"),
  openHtmlReport = TRUE,
  keepSourceFiles = FALSE,
  reportFileBaseName = "rpact_test_result_report",
  metaData = list(),
  metaDataFile = NULL,
  workers = NULL
)
```

Arguments

outDir	The absolute path to the output directory where all test results will 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; if FALSE, only a subset of tests is run.
connection	A list allowing owners of the rpact validation documentation to provide token and secret credentials for full access to unit tests, enabling them to meet regulatory requirements (see www.rpact.com for more information).
testFileDirectory	An optional path pointing to a local directory containing test files.
downloadTestsOnly	If TRUE, the unit test files are only downloaded and not executed. Default is FALSE.
addWarningDetailsToReport	If TRUE, additional warning details are included in the test report. Default is TRUE.

`reportType` The type of report to generate. Can be `"compact"`, `"detailed"`, or `"Rout"`.

`testInstalledBasicPackages`
If `TRUE`, tests for installed basic R packages are included, default is `TRUE`. For more information, see `testInstalledBasic`.

`scope` The scope of the basic R package tests to run. Can be `"basic"`, `"devel"`, `"both"`, `"internet"`, or `"all"`. Default is `"basic"`. For more information, see `testInstalledBasic`. Only available if `testInstalledBasicPackages = TRUE`.

`openHtmlReport` If `TRUE`, the HTML report is opened after the tests are completed, default is `TRUE`.

`keepSourceFiles`
If `TRUE`, the source files are kept after the tests are completed. A copy of them can be found in the subdirectory `src`.

`reportFileBaseName`
The base name for the report files (without path or extension).

`metaData` A named `list` containing additional metadata to be included in the test report. This can include information such as `company`, `container name`, `host name`, etc. Default is an empty list. For more information, see `writeKeyValueFile` and `readKeyValueFile`.

`metaDataFile` An optional path to a text file containing metadata: one entry per line in the form `KEY=VALUE`. Ignored with a warning when a non-empty `metaData` list is supplied. Supplied metadata is saved as `meta_data.txt` in the test target directory. For more information, see `writeKeyValueFile` and `readKeyValueFile`.

`workers` Number of parallel R test processes. The default `NULL` uses the available logical CPUs minus two, with a minimum of one and a maximum of 16, limited to the number of test files. Uses `parallelly::availableCores()` when installed (including resource limits), otherwise `parallel::detectCores()`. An explicit positive integer overrides the automatic selection; use `1` for serial execution. Parallel tests are supported on Windows, Linux, and macOS.

Details

This function is integral to the installation qualification (IQ) process of the `rpact` package, ensuring it meets quality standards and functions as expected. A directory named `rpact-tests` is created within the specified output directory, where all test files are downloaded from a secure resource and executed. Results are saved in the file `testthat.Rout`, located in the `rpact-tests` directory.

Unit test files are distributed across separate R processes. On Windows, standard output and error use files to avoid the pipe-related access violation in R bug 19104. Linux and macOS use the testthat parallel runner. Scheduling priorities are read from the `DESCRIPTION` above a local test directory, when present, and otherwise from the installed rpact package. Report sections follow file completion order. Personal rpact configuration files are isolated for the run. R basic package tests run in a separate R process before parallel workers start, using a temporary output directory. If execution aborts, partial results are marked as incomplete and the diagnostic log path is printed; use `workers = 1` to investigate persistent parallel-worker failures. The returned object includes `workers` and `timings`, with elapsed seconds for unit tests (including worker startup), R basic package tests, HTML rendering, and PDF rendering.

Installation qualification is a critical step in the validation process. Without successful IQ, the package cannot be considered fully validated. To gain access to the full set of unit tests, users must provide `token` and `secret` credentials, which are distributed to members of the rpact user group as part of the validation documentation. For more information, see vignette rpact_installation_qualification.

Value

Invisibly returns an InstallationQualificationResult) object.

References

For more information, please visit: https://www.rpact.org/vignettes/utilities/rpact_installation_qualification/

Examples

```
## Not run:
# Set the output directory
setwd("/path/to/output")

# Basic usage
testPackage()

# Perform all unit tests with access credentials
testPackage(
    connection = list(
        token = "your_token_here",
        secret = "your_secret_here"
    )
)

# Download test files without executing them
testPackage(downloadTestsOnly = TRUE)

## End(Not run)
```

TrialComparison *Trial Comparison*

Description

R6 container for existing trial planning objects, inheriting from ParameterSet. Create instances using getComparison().

See Also

getComparison()

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(q = time, shape = kappa, scale = 1`

/ lambda) of the stats package.
For example, getPiecewiseExponentialDistribution(time = 130, piecewiseLambda = 0.01, kappa = 4.2) and pweibull(q = 130, shape = 4.2, scale = 1 / 0.01) provide the same 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

```
## Not run:
# 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
)

## End(Not run)
```

`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)
```

`[.TrialComparisonTable` *Subset a Trial Comparison Table*

Description

Subsets a reporting data frame while preserving its rendering attributes.

Usage

```
## S3 method for class 'TrialComparisonTable'
x[i, j, ..., drop = FALSE]
```

Arguments

`x`	A `TrialComparisonTable`.
`i, j`	Row and column indices, as for a data frame.
`...`	Further arguments passed to the data-frame subsetting method.
`drop`	Whether to simplify the result, as for a data frame.

Value

A subset retaining its reporting metadata when the result is a data frame.

`[[.TrialComparison` *Access Objects in a Trial Comparison*

Description

Extracts a contained comparison object by name or position.

`$` accesses named comparison objects while preserving public R6 members.

Usage

```
## S3 method for class 'TrialComparison'
x[[i, ..., exact = TRUE]]

## S3 method for class 'TrialComparison'
x$name
```

Arguments

x	A TrialComparison.
i	A single object name or a positive integer position.
...	Further indices are not supported.
exact	Whether character indices must match exactly, as for list [[. $ always matches exactly.
name	An object name, supplied literally after $.

Details

Object names are unique. Numeric positions refer to the stored object order. R6 members take precedence for $: x$add is the method even when an object is named "add"; use x[["add"]] to select that object. [[selects contained objects first and falls back to existing R6 members for other character names. This also preserves existing internal field access. These methods provide extraction; replacing collection entries is not supported. Use getComparison() or $add() to build or extend the collection.

Value

The contained object by reference, or NULL for an unknown name.

Examples

```
comparison <- getComparison(
    fixed = getDesignFixed(),
    sequential = getDesignGroupSequential(kMax = 2)
)
identical(comparison$fixed, comparison[["fixed"]])
identical(comparison[[1]], comparison$fixed)
```

%vs% *Compare and Summarize Trial Planning Objects*

Description

Combines two operands with getComparison() and returns the summary of the resulting comparison. Repeated use extends the summarized comparison.

Usage

```
x %vs% y
```

Arguments

x, y	Supported trial planning objects, their SummaryFactory summaries, TrialComparison objects, or TrialComparisonSummary objects.

Details

Summary operands contribute their underlying `object`. For `ParameterSet` operands, the supported classes are those accepted by `getComparison()`: designs, design characteristics, sample size or power plans, simulations, and power/ASN results, event probabilities, recruitment results, conditional power, performance scores, Fisher information, and optimal conditional-error characteristics. Existing comparisons contribute their contained objects in order, retaining names and references. The operands' collection membership is not modified. The validation and unique-name rules of `getComparison()` apply.

The result can be assigned, extended with another `%vs%`, or passed directly to `plot.TrialComparison()`. Chained operations do not print intermediate summaries. Plot axes must still be specified when their selection is ambiguous.

Value

A visible `TrialComparisonSummary`, as returned by `summary(getComparison(x, y))` after unwrapping summary operands.

See Also

`getComparison()`, `compare()`

Examples

```
comparison <- getSampleSizeMeans(alternative = 0.22, stDev = 0.5) %vs%
    getSampleSizeMeans(alternative = 0.44, stDev = 1) %vs%
    getSampleSizeMeans(alternative = 0.59, stDev = 1)
comparison

comp1 <- getSampleSizeMeans(alternative = 0.22) %vs%
    compare(getSampleSizeMeans(alternative = 0.44))
comp2 <- comp1 %vs% compare(getSampleSizeMeans(alternative = 0.59))
comp2
comp2 |> getComparison()

## Not run:
getSampleSizeMeans(alternative = 0.22, stDev = 0.5) %vs%
    getSampleSizeMeans(alternative = 0.44, stDev = 1) %vs%
    getSampleSizeMeans(alternative = 0.59, stDev = 1) |>
    plot(
        xAxis = "alternative",
        yAxis = c("nFixed", "criticalValuesEffectScale"), category = "stDev"
    )

## End(Not run)
```

Index

∗ **analysis functions**
getAnalysisResults, 17
getClosedCombinationTestResults, 23
getClosedConditionalDunnettTestResults, 25
getConditionalPower, 33
getConditionalRejectionProbabilities, 36
getFinalConfidenceInterval, 62
getFinalPValue, 64
getRepeatedConfidenceIntervals, 107
getRepeatedPValues, 109
getStageResults, 196
getTestActions, 198
∗ **design functions**
getDesignCharacteristics, 44
getDesignConditionalDunnett, 45
getDesignFisher, 47
getDesignFixed, 49
getDesignGroupSequential, 51
getDesignInverseNormal, 54
getGroupSequentialProbabilities, 74
getPowerAndAverageSampleNumber, 89
∗ **output formats**
getOutputFormat, 80
setOutputFormat, 238
∗ **power functions**
getPowerCounts, 91
getPowerMeans, 94
getPowerRates, 97
getPowerSurvival, 100
∗ **sample size functions**
getSampleSizeCounts, 110
getSampleSizeMeans, 113
getSampleSizeRates, 116
getSampleSizeSurvival, 119
[.TrialComparisonTable, 251
[[.TrialComparison, 251
$.TrialComparison *([[.TrialComparison)*, 251
%vs%, 252

AccrualTime, *15*, *77*
add, 7
add(), *13*, *28*, *29*, *78*
AnalysisResults, *21*
as.data.frame(), *8*, *15*, *21*, *24*, *25*, *29*, *35*, *40*, *45*, *46*, *48*, *50*, *53*, *57*, *59*, *62*, *77*, *85*, *90*, *93*, *96*, *99*, *103*, *112*, *115*, *118*, *122*, *127*, *133*, *139*, *147*, *153*, *160*, *167*, *175*, *181*, *189*, *198*, *235*
as.data.frame.PiecewiseSurvivalTimeMultiGroup, 7
as.data.frame.TrialComparison, 8
as.data.frame.TrialComparison(), *28*, *200*, *231*, *241*
as.matrix(), *15*, *21*, *24*, *25*, *29*, *35*, *40*, *45*, *46*, *48*, *50*, *53*, *57*, *59*, *62*, *77*, *85*, *90*, *93*, *96*, *99*, *103*, *112*, *115*, *118*, *122*, *127*, *133*, *139*, *147*, *153*, *160*, *167*, *175*, *181*, *189*, *198*, *235*
as251Normal, 10
as251StudentT, 11

character, *199*, *234*
ClosedCombinationTestResults, *24*, *25*
compare, 12
compare(), *29*, *78*, *253*
ConditionalPowerResults, *35*

data.frame, *15*, *21*, *24*, *25*, *29*, *35*, *38*, *40*, *45*, *46*, *48*, *50*, *53*, *57*, *59*, *62*, *77*, *85*, *90*, *93*, *96*, *99*, *103*, *107*, *112*, *115*, *118*, *122*, *127*, *128*, *133*, *139*, *147*, *153*, *160*, *167*, *175*, *181*, *182*, *187*, *189*, *190*, *198*, *235*
Dataset, *40*, *207*, *235*, *237*
DatasetGeneral, *39*
DatasetMeans, *39*
DatasetRates, *39*
DatasetSurvival, *39*
delete, 13
delete(), *7*, *13*, *28*, *29*, *78*

EventProbabilities, *61*, *208–210*

fetch *(obtain)*, 203
fetch(), *14*, *29*, *32*, *220*
fetch.TrialComparison, 14
fetch.TrialComparison(), *13*, *27*, *29*, *203*
format, *239*

getAccrualTime, 15
getAccrualTime(), *15*, *60*, *76*, *102*, *121*, *143*, *172*, *186*
getAnalysisResults, 17
getAnalysisResults(), *24*, *26*, *35*, *36*, *40*, *64*, *65*, *79*, *109*, *110*, *198*, *199*, *204*, *216*, *237*, *238*
getAvailablePlotTypes *(plotTypes)*, 228
getClosedCombinationTestResults, 23
getClosedCombinationTestResults(), *21*, *26*, *35*, *36*, *64*, *65*, *109*, *110*, *198*, *199*
getClosedConditionalDunnettTestResults, 25
getClosedConditionalDunnettTestResults(), *21*, *24*, *35*, *36*, *46*, *64*, *65*, *109*, *110*, *198*, *199*
getComparison, 26
getComparison(), *7*, *10*, *12–14*, *49*, *59*, *78*, *241*, *245*, *252*, *253*
getComparisonParameterDescriptions, 30
getComparisonParameterDescriptions(), *231*
getComparisonParameterNames, 31
getComparisonPlotParameters, 32
getConditionalPower, 33
getConditionalPower(), *21*, *24*, *26*, *36*, *64*, *65*, *109*, *110*, *198*, *199*
getConditionalRejectionProbabilities, 36
getConditionalRejectionProbabilities(), *21*, *24*, *26*, *35*, *64*, *65*, *109*, *110*, *198*, *199*
getData, 37
getData(), *106*, *107*, *128*, *153*, *182*, *190*
getDataSet *(getDataset)*, 38
getDataset, 38
getDataset(), *18*, *63*, *108*, *196*, *235*, *237*
getDesignCharacteristics, 44
getDesignCharacteristics(), *47*, *49*, *50*, *54*, *58*, *75*, *90*
getDesignConditionalDunnett, 45
getDesignConditionalDunnett(), *25*, *45*, *49*, *50*, *54*, *58*, *75*, *90*, *130*, *136*, *142*, *156*, *164*, *170*
getDesignFisher, 47
getDesignFisher(), *45*, *47*, *50*, *54*, *58*, *73*, *75*, *90*, *130*, *136*, *142*, *156*, *164*, *170*, *221*
getDesignFixed, 49
getDesignFixed(), *45*, *47*, *49*, *54*, *58*, *75*, *90*, *130*, *136*, *142*, *156*, *164*, *170*
getDesignGroupSequential, 51
getDesignGroupSequential(), *45*, *47*, *49*, *50*, *58*, *73*, *75*, *90*, *130*, *136*, *142*, *156*, *164*, *170*, *221*, *229*
getDesignInverseNormal, 54
getDesignInverseNormal(), *45*, *47*, *49*, *50*, *54*, *73*, *75*, *90*, *130*, *136*, *142*, *156*, *164*, *170*, *221*
getDesignSet, 58
getDesignSet(), *54*, *58*, *226*
getEventProbabilities, 60
getFinalConfidenceInterval, 62
getFinalConfidenceInterval(), *21*, *24*, *26*, *35*, *36*, *65*, *109*, *110*, *198*, *199*
getFinalPValue, 64
getFinalPValue(), *21*, *24*, *26*, *35*, *36*, *64*, *109*, *110*, *198*, *199*
getFisherInformation, 65
getFisherInformation(), *69*, *71–73*
getFutilityBounds, 68
getFutilityBounds(), *49*, *54*, *65*, *68*, *238*
getGroupSequentialProbabilities, 74
getGroupSequentialProbabilities(), *45*, *47*, *49*, *50*, *54*, *58*, *90*
getNumberOfSubjects, 76
getNumberOfSubjects(), *16*
getObjectRCode *(rcmd)*, 232
getObjectRCode(), *28*, *234*
getObjects, 78
getObjects(), *13*, *27*, *29*
getObservedInformationRates, 79
getObservedInformationRates(), *21*
getOutputFormat, 80
getOutputFormat(), *239*
getPerformanceScore, 82
getPiecewiseExponentialDistribution *(utilitiesForPiecewiseExponentialDistribution)*, 246
getPiecewiseExponentialQuantile *(utilitiesForPiecewiseExponentialDistribution)*, 246
getPiecewiseExponentialRandomNumbers *(utilitiesForPiecewiseExponentialDistribution)*, 246
getPiecewiseHazardRatioArray, 83
getPiecewiseSurvivalTime, 84
getPiecewiseSurvivalTime(), *61*, *102*, *121*, *186*

getPiecewiseSurvivalTimeEnrichment, 86
getPiecewiseSurvivalTimeMultiArm, 88
getPlotSettings(), *206, 207*, *210, 211*, *213*, *215*, *217*, *222*, *225*, *227*
getPowerAndAverageSampleNumber, 89
getPowerAndAverageSampleNumber(), *28*, *45*, *47*, *49, 50*, *54*, *58*, *75*, *222*
getPowerCounts, 91
getPowerCounts(), *66*, *96*, *99*, *104*, *224*
getPowerMeans, 94
getPowerMeans(), *66*, *93*, *99*, *104*, *224*
getPowerRates, 97
getPowerRates(), *66*, *93*, *96*, *104*, *224*
getPowerSurvival, 100
getPowerSurvival(), *66*, *93*, *96*, *99*, *224*
getRawData, 106
getRawData(), *145*, *174*, *187*, *190*
getRepeatedConfidenceIntervals, 107
getRepeatedConfidenceIntervals(), *21*, *24*, *26*, *35, 36*, *64, 65*, *110*, *198, 199*
getRepeatedPValues, 109
getRepeatedPValues(), *21*, *24*, *26*, *35, 36*, *64, 65*, *109*, *198, 199*
getSampleSizeCounts, 110
getSampleSizeCounts(), *66*, *115*, *118*, *123*, *224*
getSampleSizeMeans, 113
getSampleSizeMeans(), *50*, *52*, *55*, *66*, *113*, *118*, *123*, *223*, *229*
getSampleSizeRates, 116
getSampleSizeRates(), *66*, *113*, *115*, *123*, *223*
getSampleSizeSurvival, 119
getSampleSizeSurvival(), *61*, *66*, *77*, *113*, *115*, *118*, *223*
getSimulationCounts, 125
getSimulationCounts(), *66*
getSimulationEnrichmentMeans, 129
getSimulationEnrichmentRates, 135
getSimulationEnrichmentSurvival, 141
getSimulationEnrichmentSurvival(), *106*
getSimulationMeans, 150
getSimulationMeans(), *37, 38*, *66*
getSimulationMultiArmMeans, 155
getSimulationMultiArmMeans(), *37, 38*
getSimulationMultiArmRates, 163
getSimulationMultiArmRates(), *37, 38*
getSimulationMultiArmSurvival, 169
getSimulationMultiArmSurvival(), *37, 38*, *106*
getSimulationRates, 178
getSimulationRates(), *37, 38*, *66*
getSimulationSurvival, 184
getSimulationSurvival(), *37*, *66*, *106*, *214*
getStageResults, 196
getStageResults(), *21*, *24–26*, *34–36*, *64, 65*, *109, 110*, *198, 199*, *216*
getTestActions, 198
getTestActions(), *21*, *24*, *26*, *35, 36*, *64, 65*, *109, 110*, *198*

InstallationQualificationResult, *245*

knit_print, *199*
knit_print.SummaryFactory, 199
knit_print.TrialComparisonTable, 200

length, *59*
length(), *29*
length.TrialComparison *(*getComparison*)*, 26
list, *64, 65*, *237*

make.names, *9*
matrix, *15*, *21*, *24, 25*, *29*, *35, 36*, *40*, *45, 46*, *48*, *50*, *53*, *57*, *59*, *62*, *77*, *85*, *90*, *93*, *96*, *99*, *103*, *109, 110*, *112*, *115*, *118*, *122*, *127*, *133*, *139*, *147*, *153*, *160*, *167*, *175*, *181*, *189*, *198*, *235*
methods, *16*, *21*, *24*, *26*, *29*, *35*, *45*, *47*, *49*, *54*, *57*, *59*, *62*, *77*, *85*, *90*, *93*, *96*, *99*, *104*, *113*, *115*, *118*, *123*, *128*, *134*, *140*, *148*, *154*, *161*, *168*, *177*, *182*, *191*, *198*
mvnprd, *11*, 201
mvstud, *12*, 202

names, *21*, *59*, *84*, *197*
names(), *15*, *24, 25*, *29*, *35*, *40*, *44*, *46*, *48*, *50*, *53*, *57*, *62*, *77*, *85*, *90*, *93*, *96*, *99*, *103*, *112*, *115*, *118*, *122*, *127*, *133*, *139*, *147*, *153*, *160*, *167*, *175*, *181*, *188*, *235*
names.TrialComparison *(*getComparison*)*, 26
nMax, *222*
NumberOfSubjects, *77*, *209, 210*
numeric, *36*, *110*, *199*, *247*

obtain, 203

ParameterSet, *203*, *212*
PiecewiseSurvivalTime, *85*
plot arguments, *204*, *216*
plot(), *15*, *21*, *24, 25*, *29*, *35*, *40*, *45, 46*, *48*, *50*, *53*, *57*, *59*, *62*, *77*, *85*, *90*, *93*, *96*,

99, *103*, *112*, *115*, *118*, *122*, *127*, *133*, *139*, *147*, *153*, *160*, *167*, *175*, *181*, *188*, *198*, *223*, *235*
plot.AnalysisResults, 204
plot.AnalysisResults(), *35*
plot.Dataset, 206
plot.EventProbabilities, 208
plot.NumberOfSubjects, 210
plot.ParameterSet, 212
plot.SimulationResults, 213
plot.StageResults, 216
plot.StageResults(), *35*
plot.SummaryFactory, 218
plot.TrialComparison, 219
plot.TrialComparison(), *28*, *32*, *33*, *219*, *253*
plot.TrialComparisonSummary *(*plot.TrialComparison*)*, 219
plot.TrialDesign, 220
plot.TrialDesignCharacteristics *(*plot.TrialDesign*)*, 220
plot.TrialDesignPlan, 223
plot.TrialDesignSet, 225
plot.TrialDesignSummaries, 228
plotTypes, 228
PowerAndAverageSampleNumberResult, *90*
ppwexp *(*utilitiesForPiecewiseExponentialDistribution*)*, 246
print, *206*, *215*, *222*, *225*, *227*, *228*
print(), *15*, *21*, *24*, *25*, *29*, *35*, *40*, *44*, *46*, *48*, *50*, *53*, *57*, *59*, *62*, *77*, *85*, *90*, *93*, *96*, *99*, *103*, *112*, *115*, *118*, *122*, *127*, *133*, *139*, *147*, *153*, *160*, *167*, *175*, *181*, *188*, *198*, *235*
print.SummaryFactory, 230
print.TrialComparison, 230
print.TrialDesignCharacteristics, 231
print.TrialDesignSummaries, 232

qpwexp *(*utilitiesForPiecewiseExponentialDistribution*)*, 246

range, *128*, *153*, *181*, *189*
rcmd, 232
rcmd(), *234*
read.table, *234–236*
readDataset, 234
readDataset(), *237*, *249*, *250*
readDatasets, 236
readDatasets(), *235*, *249*, *250*
readKeyValueFile, *244*
reshape, *235*
rpact, 237
rpact-package *(*rpact*)*, 237
rpwexp *(*utilitiesForPiecewiseExponentialDistribution*)*, 246

setOutputFormat, 238
setOutputFormat(), *81*
SimulationResults, *37*, *106*, *127*, *132*, *139*, *147*, *153*, *160*, *167*, *175*, *181*, *188*
SimulationResultsEnrichmentSurvival, *147*, *175*, *188*
SimulationResultsMultiArmSurvival, *147*, *175*, *188*
SimulationResultsSurvival, *147*, *175*, *188*
StageResults, *197*
summary(), *15*, *21*, *24*, *25*, *29*, *35*, *40*, *44*, *46*, *48*, *50*, *53*, *57*, *59*, *62*, *77*, *85*, *90*, *93*, *96*, *99*, *103*, *112*, *115*, *118*, *122*, *127*, *133*, *139*, *147*, *153*, *160*, *167*, *175*, *181*, *188*, *198*, *235*
summary.TrialComparison, 240
summary.TrialComparison(), *10*
summary.TrialComparisonSummary *(*summary.TrialComparison*)*, 240

testInstalledBasic, *244*
testPackage, 243
testPackage(), *238*
thetaH0, *204*, *216*
TrialComparison, *28*, 245
TrialDesign, *46*, *48*, *50*, *53*, *57*
TrialDesignCharacteristics, *44*
TrialDesignPlan, *93*, *96*, *99*, *103*, *112*, *115*, *118*, *122*
TrialDesignSet, *59*

utilitiesForPiecewiseExponentialDistribution, 246

write.table, *248–250*
writeDataset, 248
writeDataset(), *235*, *237*, *249*, *250*
writeDatasets, 249
writeDatasets(), *235*, *237*, *249*
writeKeyValueFile, *244*