Package: gsDesign2 (via r-universe)

July 1, 2026

Title Group Sequential Design with Non-Constant Effect

Version 1.1.9

Description The goal of 'gsDesign2' is to enable fixed or group sequential design under non-proportional hazards. To enable highly flexible enrollment, time-to-event and time-to-dropout assumptions, 'gsDesign2' offers piecewise constant enrollment, failure rates, and dropout rates for a stratified population. This package includes three methods for designs: average hazard ratio, weighted logrank tests in Yung and Liu (2019) <doi:10.1111/biom.13196>, and MaxCombo tests. Substantial flexibility on top of what is in the 'gsDesign' package is intended for selecting boundaries.

License GPL-3

URL `https://merck.github.io/gsDesign2/`, `https://github.com/Merck/gsDesign2`

BugReports `https://github.com/Merck/gsDesign2/issues`

Encoding UTF-8

Depends R (>= 4.1.0)

Imports corpcor, data.table, dplyr, gsDesign, gt, methods, mvtnorm, npsurvSS (>= 1.1.0), r2rtf, stats, survival, tibble, tidyr, utils, Rcpp

Suggests covr, ggplot2, cowplot, kableExtra, knitr, rmarkdown, simtrial, testit (>= 1.0)

VignetteBuilder knitr

LinkingTo Rcpp

RoxygenNote 7.3.3

NeedsCompilation yes

Author Keaven Anderson [aut], Yujie Zhao [aut, cre], Yilong Zhang [aut], John Blischak [aut], Yihui Xie [aut], Nan Xiao [aut], Jianxiao Yang [aut], Amin Shirazi [ctb], Ruixue Wang [ctb], Yi Cui [ctb], Ping Yang [ctb], Xin Tong Li [ctb], Chenxiang Li

[ctb], Hiroaki Fukuda [ctb], Hongtao Zhang [ctb], Yalin Zhu [ctb], Shiyu Zhang [ctb], Dickson Wanjau [ctb], Merck & Co., Inc., Rahway, NJ, USA and its affiliates [cph] (ROR: <https://ror.org/02891sr49>)

Maintainer Yujie Zhao <yujie.zhao@merck.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-06-30 22:40:02 UTC

RemoteUrl https://github.com/cran/gsDesign2

RemoteRef HEAD

RemoteSha 183cd3792f8b424d790552d7dea2b8d405c850d6

Contents

`ahr` *Average hazard ratio under non-proportional hazards*

Description

Provides a geometric average hazard ratio under various non-proportional hazards assumptions for either single or multiple strata studies. The piecewise exponential distribution allows a simple method to specify a distribution and enrollment pattern where the enrollment, failure and dropout rates changes over time.

Usage

```
ahr(
  enroll_rate = define_enroll_rate(duration = c(2, 2, 10), rate = c(3, 6, 9)),
  fail_rate = define_fail_rate(duration = c(3, 100), fail_rate = log(2)/c(9, 18), hr =
    c(0.9, 0.6), dropout_rate = 0.001),
  total_duration = 30,
  ratio = 1
)
```

Arguments

`enroll_rate`	An `enroll_rate` data frame with or without stratum created by `define_enroll_rate()`.
`fail_rate`	A `fail_rate` data frame with or without stratum created by `define_fail_rate()`.
`total_duration`	Total follow-up from start of enrollment to data cutoff; this can be a single value or a vector of positive numbers.
`ratio`	Experimental:Control randomization ratio.

Value

A data frame with `time` (from `total_duration`), `ahr` (average hazard ratio), `n` (sample size), `event` (expected number of events), `info` (information under given scenarios), and info0 (information under related null hypothesis) for each value of `total_duration` input.

Specification

- Validate if input enrollment rate contains stratum column.
- Validate if input enrollment rate contains total duration column.
- Validate if input enrollment rate contains rate column.
- Validate if input failure rate contains stratum column.
- Validate if input failure rate contains duration column.
- Validate if input failure rate contains failure rate column.
- Validate if input failure rate contains hazard ratio column.
- Validate if input failure rate contains dropout rate column.
- Validate if input trial total follow-up (total duration) is a non-empty vector of positive integers.
- Validate if strata is the same in enrollment rate and failure rate.
- Compute the proportion in each group.
- Compute the expected events by treatment groups, stratum and time period.
- Calculate the expected number of events for all time points in the total duration and for all stratification variables.
  - Compute the expected events in for each strata.
    - Combine the expected number of events of all stratification variables.
    - Recompute events, hazard ratio and information under the given scenario of the combined data for each strata.
  - Combine the results for all time points by summarizing the results by adding up the number of events, information under the null and the given scenarios.
- Return a data frame of overall event count, statistical information and average hazard ratio of each value in total_duration.
- Calculation of `ahr` for different design scenarios, and the comparison to the simulation studies are defined in vignette/AHRVignette.Rmd.

Examples

```
# Example 1: default
ahr()

# Example 2: default with multiple analysis times (varying total_duration)
ahr(total_duration = c(15, 30))

# Example 3: stratified population
enroll_rate <- define_enroll_rate(
  stratum = c(rep("Low", 2), rep("High", 3)),
  duration = c(2, 10, 4, 4, 8),
  rate = c(5, 10, 0, 3, 6)
)
fail_rate <- define_fail_rate(
  stratum = c(rep("Low", 2), rep("High", 2)),
  duration = c(1, Inf, 1, Inf),
  fail_rate = c(.1, .2, .3, .4),
```

```
  dropout_rate = .001,
  hr = c(.9, .75, .8, .6)
)
ahr(enroll_rate = enroll_rate, fail_rate = fail_rate, total_duration = c(15, 30))
```

`ahr_blinded` *Blinded estimation of average hazard ratio*

Description

Based on blinded data and assumed hazard ratios in different intervals, compute a blinded estimate of average hazard ratio (AHR) and corresponding estimate of statistical information. This function is intended for use in computing futility bounds based on spending assuming the input hazard ratio (hr) values for intervals specified here.

Usage

```
ahr_blinded(
  surv = survival::Surv(time = simtrial::ex1_delayed_effect$month, event =
    simtrial::ex1_delayed_effect$evntd),
  intervals = c(3, Inf),
  hr = c(1, 0.6),
  ratio = 1
)
```

Arguments

`surv`	Input survival object (see `survival::Surv()`); note that only 0 = censored, 1 = event for `survival::Surv()`.
`intervals`	Vector containing positive values indicating interval lengths where the exponential rates are assumed. Note that a final infinite interval is added if any events occur after the final interval specified.
`hr`	Vector of hazard ratios assumed for each interval.
`ratio`	Ratio of experimental to control randomization.

Value

A `tibble` with one row containing

- `ahr` - Blinded average hazard ratio based on assumed period-specific hazard ratios input in `fail_rate` and observed events in the corresponding intervals.
- `event` - Total observed number of events.
- `info0` - Information under related null hypothesis.
- `theta` - Natural parameter for group sequential design representing expected incremental drift at all analyses.

Specification

- Validate input hr is a numeric vector.
- Validate input hr is non-negative.
- Validate input intervals is a numeric vector > 0.
- Set final value in intervals to Inf
- Validate that hr and intervals have same length.
- For input time-to-event data, count number of events in each input interval by stratum.
- Compute the blinded estimate of average hazard ratio.
- Compute adjustment for information.
- Return a tibble of the sum of events, average hazard ratio, blinded average hazard ratio, and the information.

Examples

```
ahr_blinded(
  surv = survival::Surv(
    time = simtrial::ex2_delayed_effect$month,
    event = simtrial::ex2_delayed_effect$evntd
  ),
  intervals = c(4, 100),
  hr = c(1, .55),
  ratio = 1
)
```

---

as_gt — *Convert summary table of a fixed or group sequential design object to a gt object*

Description

Convert summary table of a fixed or group sequential design object to a gt object

Usage

```
as_gt(x, ...)

## S3 method for class 'fixed_design_summary'
as_gt(x, title = NULL, footnote = NULL, ...)

## S3 method for class 'gs_design_summary'
as_gt(
  x,
  title = NULL,
  subtitle = NULL,
  colname_spanner = "Cumulative boundary crossing probability",
```

```
  colname_spannersub = c("Alternate hypothesis", "Null hypothesis"),
  footnote = NULL,
  display_bound = c("Efficacy", "Futility"),
  display_columns = NULL,
  display_inf_bound = FALSE,
  ...
)
```

Arguments

- `x` — A summary object of a fixed or group sequential design.
- `...` — Additional arguments (not used).
- `title` — A string to specify the title of the gt table.
- `footnote` — A list containing `content`, `location`, and `attr`. `content` is a vector of string to specify the footnote text; `location` is a vector of string to specify the locations to put the superscript of the footnote index; `attr` is a vector of string to specify the attributes of the footnotes, for example, `c("colname", "title", "subtitle", "analysis", "spanner")`; users can use the functions in the `gt` package to customize the table. To disable footnotes, use `footnote = FALSE`.
- `subtitle` — A string to specify the subtitle of the gt table.
- `colname_spanner` — A string to specify the spanner of the gt table.
- `colname_spannersub` — A vector of strings to specify the spanner details of the gt table.
- `display_bound` — A vector of strings specifying the label of the bounds. The default is `c("Efficacy", "Futility")`.
- `display_columns` — A vector of strings specifying the variables to be displayed in the summary table.
- `display_inf_bound` — Logical, whether to display the +/-inf bound.

Value

A `gt_tbl` object.

Examples

```
# Fixed design examples ----

# Enrollment rate
enroll_rate <- define_enroll_rate(
  duration = 18,
  rate = 20
)

# Failure rates
fail_rate <- define_fail_rate(
  duration = c(4, 100),
```

```
  fail_rate = log(2) / 12,
  dropout_rate = .001,
  hr = c(1, .6)
)

# Study duration in months
study_duration <- 36

# Experimental / Control randomization ratio
ratio <- 1

# 1-sided Type I error
alpha <- 0.025

# Type II error (1 - power)
beta <- 0.1

# Example 1 ----
fixed_design_ahr(
  alpha = alpha, power = 1 - beta,
  enroll_rate = enroll_rate, fail_rate = fail_rate,
  study_duration = study_duration, ratio = ratio
) |>
  summary() |>
  as_gt()

# Example 2 ----
fixed_design_fh(
  alpha = alpha, power = 1 - beta,
  enroll_rate = enroll_rate, fail_rate = fail_rate,
  study_duration = study_duration, ratio = ratio
) |>
  summary() |>
  as_gt()

# Group sequential design examples ---

# Example 1 ----
# The default output

gs_design_ahr() |>
  summary() |>
  as_gt()

gs_power_ahr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_gt()

gs_design_wlr() |>
  summary() |>
  as_gt()

gs_power_wlr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
```

```
  summary() |>
  as_gt()

gs_power_combo() |>
  summary() |>
  as_gt()

gs_design_rd() |>
  summary() |>
  as_gt()

gs_power_rd() |>
  summary() |>
  as_gt()

# Example 2 ----
# Usage of title = ..., subtitle = ...
# to edit the title/subtitle
gs_power_wlr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_gt(
    title = "Bound Summary",
    subtitle = "from gs_power_wlr"
  )

# Example 3 ----
# Usage of colname_spanner = ..., colname_spannersub = ...
# to edit the spanner and its sub-spanner
gs_power_wlr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_gt(
    colname_spanner = "Cumulative probability to cross boundaries",
    colname_spannersub = c("under H1", "under H0")
  )

# Example 4 ----
# Usage of footnote = ...
# to edit the footnote
gs_power_wlr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_gt(
    footnote = list(
      content = c(
        "approximate weighted hazard ratio to cross bound.",
        "wAHR is the weighted AHR.",
        "the crossing probability.",
        "this table is generated by gs_power_wlr."
      ),
      location = c("~wHR at bound", NA, NA, NA),
      attr = c("colname", "analysis", "spanner", "title")
    )
  )
```

```
# Example 5 ----
# Usage of display_bound = ...
# to either show efficacy bound or futility bound, or both(default)
gs_power_wlr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_gt(display_bound = "Efficacy")

# Example 6 ----
# Usage of display_columns = ...
# to select the columns to display in the summary table
gs_power_wlr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_gt(display_columns = c("Analysis", "Bound", "Nominal p", "Z", "Probability"))
```

`as_rtf` *Write summary table of a fixed or group sequential design object to an RTF file*

Description

Write summary table of a fixed or group sequential design object to an RTF file

Usage

```
as_rtf(x, ...)

## S3 method for class 'fixed_design_summary'
as_rtf(
  x,
  title = NULL,
  footnote = NULL,
  col_rel_width = NULL,
  orientation = c("portrait", "landscape"),
  text_font_size = 9,
  file,
  ...
)

## S3 method for class 'gs_design_summary'
as_rtf(
  x,
  title = NULL,
  subtitle = NULL,
  colname_spanner = "Cumulative boundary crossing probability",
  colname_spannersub = c("Alternate hypothesis", "Null hypothesis"),
  footnote = NULL,
  display_bound = c("Efficacy", "Futility"),
```

```
    display_columns = NULL,
    display_inf_bound = TRUE,
    col_rel_width = NULL,
    orientation = c("portrait", "landscape"),
    text_font_size = 9,
    file,
    ...
)
```

Arguments

`x` A summary object of a fixed or group sequential design.

`...` Additional arguments (not used).

`title` A string to specify the title of the RTF table.

`footnote` A list containing `content`, `location`, and `attr`. `content` is a vector of string to specify the footnote text; `location` is a vector of string to specify the locations to put the superscript of the footnote index; `attr` is a vector of string to specify the attributes of the footnotes, for example, `c("colname", "title", "subtitle", "analysis", "spanner")`; users can use the functions in the `gt` package to customize the table.

`col_rel_width` Column relative width in a vector e.g. c(2,1,1) refers to 2:1:1. Default is NULL for equal column width.

`orientation` Orientation in 'portrait' or 'landscape'.

`text_font_size` Text font size. To vary text font size by column, use numeric vector with length of vector equal to number of columns displayed e.g. c(9,20,40).

`file` File path for the output.

`subtitle` A string to specify the subtitle of the RTF table.

`colname_spanner` A string to specify the spanner of the RTF table.

`colname_spannersub` A vector of strings to specify the spanner details of the RTF table.

`display_bound` A vector of strings specifying the label of the bounds. The default is `c("Efficacy", "Futility")`.

`display_columns` A vector of strings specifying the variables to be displayed in the summary table.

`display_inf_bound` Logical, whether to display the +/-inf bound.

Value

`as_rtf()` returns the input `x` invisibly.

Examples

```
# Enrollment rate
enroll_rate <- define_enroll_rate(
  duration = 18,
  rate = 20
)

# Failure rates
fail_rate <- define_fail_rate(
  duration = c(4, 100),
  fail_rate = log(2) / 12,
  dropout_rate = .001,
  hr = c(1, .6)
)

# Study duration in months
study_duration <- 36

# Experimental / Control randomization ratio
ratio <- 1

# 1-sided Type I error
alpha <- 0.025

# Type II error (1 - power)
beta <- 0.1

# AHR ----
# under fixed power
x <- fixed_design_ahr(
  alpha = alpha, power = 1 - beta,
  enroll_rate = enroll_rate, fail_rate = fail_rate,
  study_duration = study_duration, ratio = ratio
) |> summary()
x |> as_rtf(file = tempfile(fileext = ".rtf"))
x |> as_rtf(title = "Fixed design", file = tempfile(fileext = ".rtf"))
x |> as_rtf(
  footnote = "Power computed with average hazard ratio method given the sample size",
  file = tempfile(fileext = ".rtf")
)
x |> as_rtf(text_font_size = 10, file = tempfile(fileext = ".rtf"))

# FH ----
# under fixed power
fixed_design_fh(
  alpha = alpha, power = 1 - beta,
  enroll_rate = enroll_rate, fail_rate = fail_rate,
  study_duration = study_duration, ratio = ratio
) |>
  summary() |>
  as_rtf(file = tempfile(fileext = ".rtf"))
#'
```

```
# the default output

gs_design_ahr() |>
  summary() |>
  as_rtf(file = tempfile(fileext = ".rtf"))

gs_power_ahr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_rtf(file = tempfile(fileext = ".rtf"))

gs_design_wlr() |>
  summary() |>
  as_rtf(file = tempfile(fileext = ".rtf"))

gs_power_wlr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_rtf(file = tempfile(fileext = ".rtf"))


gs_power_combo() |>
  summary() |>
  as_rtf(file = tempfile(fileext = ".rtf"))

gs_design_rd() |>
  summary() |>
  as_rtf(file = tempfile(fileext = ".rtf"))

gs_power_rd() |>
  summary() |>
  as_rtf(file = tempfile(fileext = ".rtf"))

# usage of title = ..., subtitle = ...
# to edit the title/subtitle
gs_power_wlr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_rtf(
    title = "Bound Summary",
    subtitle = "from gs_power_wlr",
    file = tempfile(fileext = ".rtf")
  )

# usage of colname_spanner = ..., colname_spannersub = ...
# to edit the spanner and its sub-spanner
gs_power_wlr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_rtf(
    colname_spanner = "Cumulative probability to cross boundaries",
    colname_spannersub = c("under H1", "under H0"),
    file = tempfile(fileext = ".rtf")
  )

# usage of footnote = ...
```

```
# to edit the footnote
gs_power_wlr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_rtf(
    footnote = list(
      content = c(
        "approximate weighted hazard ratio to cross bound.",
        "wAHR is the weighted AHR.",
        "the crossing probability.",
        "this table is generated by gs_power_wlr."
      ),
      location = c("~wHR at bound", NA, NA, NA),
      attr = c("colname", "analysis", "spanner", "title")
    ),
    file = tempfile(fileext = ".rtf")
  )

# usage of display_bound = ...
# to either show efficacy bound or futility bound, or both(default)
gs_power_wlr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_rtf(
    display_bound = "Efficacy",
    file = tempfile(fileext = ".rtf")
  )

# usage of display_columns = ...
# to select the columns to display in the summary table
gs_power_wlr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1)) |>
  summary() |>
  as_rtf(
    display_columns = c("Analysis", "Bound", "Nominal p", "Z", "Probability"),
    file = tempfile(fileext = ".rtf")
  )
```

`define_enroll_rate` *Define enrollment rate*

Description

Define the enrollment rate of subjects for a study as following a piecewise exponential distribution.

Usage

```
define_enroll_rate(duration, rate, stratum = "All")
```

Arguments

`duration`	A numeric vector of ordered piecewise study duration interval.
`rate`	A numeric vector of enrollment rate in each `duration`.
`stratum`	A character vector of stratum name.

Details

The `duration` are ordered piecewise for a duration equal to $t_i - t_{i-1}$, where $0 = t_0 < t_i < \cdots < t_M = \infty$. The enrollment rates are defined in each duration with the same length.

For a study with multiple strata, different duration and rates can be specified in each stratum.

Value

An `enroll_rate` data frame.

Examples

```
# Define enroll rate without stratum
define_enroll_rate(
  duration = c(2, 2, 10),
  rate = c(3, 6, 9)
)

# Define enroll rate with stratum
define_enroll_rate(
  duration = rep(c(2, 2, 2, 18), 3),
  rate = c((1:4) / 3, (1:4) / 2, (1:4) / 6),
  stratum = c(array("High", 4), array("Moderate", 4), array("Low", 4))
)
```

`define_fail_rate` *Define failure rate*

Description

Define subject failure rate for a study with two treatment groups. Also supports stratified designs that have different failure rates in each stratum.

Usage

```
define_fail_rate(duration, fail_rate, dropout_rate, hr = 1, stratum = "All")
```

Arguments

`duration`	A numeric vector of ordered piecewise study duration interval.
`fail_rate`	A numeric vector of failure rate in each `duration` in the control group.
`dropout_rate`	A numeric vector of dropout rate in each `duration`.
`hr`	A numeric vector of hazard ratio between treatment and control group.
`stratum`	A character vector of stratum name.

Details

Define the failure and dropout rate of subjects for a study as following a piecewise exponential distribution. The `duration` are ordered piecewise for a duration equal to $t_i - t_{i-1}$, where $0 = t_0 < t_i < \cdots < t_M = \infty$. The failure rate, dropout rate, and hazard ratio in a study duration can be specified.

For a study with multiple strata, different duration, failure rates, dropout rates, and hazard ratios can be specified in each stratum.

Value

A `fail_rate` data frame.

Examples

```
# Define enroll rate
define_fail_rate(
  duration = c(3, 100),
  fail_rate = log(2) / c(9, 18),
  hr = c(.9, .6),
  dropout_rate = .001
)

# Define enroll rate with stratum
define_fail_rate(
  stratum = c(rep("Low", 2), rep("High", 2)),
  duration = 1,
  fail_rate = c(.1, .2, .3, .4),
  dropout_rate = .001,
  hr = c(.9, .75, .8, .6)
)
```

`expected_accrual` *Piecewise constant expected accrual*

Description

Computes the expected cumulative enrollment (accrual) given a set of piecewise constant enrollment rates and times.

Usage

```
expected_accrual(
  time = 0:24,
  enroll_rate = define_enroll_rate(duration = c(3, 3, 18), rate = c(5, 10, 20))
)
```

Arguments

`time`	Times at which enrollment is to be computed.
`enroll_rate`	An `enroll_rate` data frame with or without stratum created by `define_enroll_rate()`.

Value

A vector with expected cumulative enrollment for the specified `times`.

Specification

- Validate if input x is a vector of strictly increasing non-negative numeric elements.
- Validate if input enrollment rate is of type data.frame.
- Validate if input enrollment rate contains duration column.
- Validate if input enrollment rate contains rate column.
- Validate if rate in input enrollment rate is non-negative with at least one positive rate.
- Convert rates to step function.
- Add times where rates change to enrollment rates.
- Make a tibble of the input time points x, duration, enrollment rates at points, and expected accrual.
- Extract the expected cumulative or survival enrollment.
- Return `expected_accrual`

Examples

```
library(tibble)

# Example 1: default
expected_accrual()

# Example 2: unstratified design
expected_accrual(
  time = c(5, 10, 20),
  enroll_rate = define_enroll_rate(
    duration = c(3, 3, 18),
    rate = c(5, 10, 20)
  )
)

# Example 3: stratified design
expected_accrual(
  time = c(24, 30, 40),
  enroll_rate = define_enroll_rate(
    stratum = c("subgroup", "complement"),
    duration = c(33, 33),
    rate = c(30, 30)
  )
)
```

```
# Example 4: expected accrual over time
# Scenario 4.1: for the enrollment in the first 3 months,
# it is exactly 3 * 5 = 15.
expected_accrual(
  time = 3,
  enroll_rate = define_enroll_rate(duration = c(3, 3, 18), rate = c(5, 10, 20))
)

# Scenario 4.2: for the enrollment in the first 6 months,
# it is exactly 3 * 5 + 3 * 10 = 45.
expected_accrual(
  time = 6,
  enroll_rate = define_enroll_rate(duration = c(3, 3, 18), rate = c(5, 10, 20))
)

# Scenario 4.3: for the enrollment in the first 24 months,
# it is exactly 3 * 5 + 3 * 10 + 18 * 20 = 405.
expected_accrual(
  time = 24,
  enroll_rate = define_enroll_rate(duration = c(3, 3, 18), rate = c(5, 10, 20))
)

# Scenario 4.4: for the enrollment after 24 months,
# it is the same as that from the 24 months, since the enrollment is stopped.
expected_accrual(
  time = 25,
  enroll_rate = define_enroll_rate(duration = c(3, 3, 18), rate = c(5, 10, 20))
)

# Instead of compute the enrolled subjects one time point by one time point,
# we can also compute it once.
expected_accrual(
  time = c(3, 6, 24, 25),
  enroll_rate = define_enroll_rate(duration = c(3, 3, 18), rate = c(5, 10, 20))
)
```

expected_event *Expected events observed under piecewise exponential model*

Description

Computes expected events over time and by strata under the assumption of piecewise constant enrollment rates and piecewise exponential failure and censoring rates. The piecewise exponential distribution allows a simple method to specify a distribution and enrollment pattern where the enrollment, failure and dropout rates changes over time. While the main purpose may be to generate a trial that can be analyzed at a single point in time or using group sequential methods, the routine can also be used to simulate an adaptive trial design. The intent is to enable sample size calculations under non-proportional hazards assumptions for stratified populations.

Usage

```
expected_event(
  enroll_rate = define_enroll_rate(duration = c(2, 2, 10), rate = c(3, 6, 9)),
  fail_rate = define_fail_rate(duration = c(3, 100), fail_rate = log(2)/c(9, 18),
    dropout_rate = 0.001),
  total_duration = 25,
  simple = TRUE
)
```

Arguments

enroll_rate	An enroll_rate data frame with or without stratum created by define_enroll_rate().
fail_rate	A fail_rate data frame with or without stratum created by define_fail_rate().
total_duration	Total follow-up from start of enrollment to data cutoff.
simple	If default (TRUE), return numeric expected number of events, otherwise a data frame as described below.

Details

More periods will generally be supplied in output than those that are input. The intent is to enable expected event calculations in a tidy format to maximize flexibility for a variety of purposes.

Value

The default when simple = TRUE is to return the total expected number of events as a real number. Otherwise, when simple = FALSE, a data frame is returned with the following variables for each period specified in fail_rate:

- t: start of period.
- fail_rate: failure rate during the period.
- event: expected events during the period.

The records in the returned data frame correspond to the input data frame fail_rate.

Specification

- Validate if input enrollment rate contains total duration column.
- Validate if input enrollment rate contains rate column.
- Validate if input failure rate contains duration column.
- Validate if input failure rate contains failure rate column.
- Validate if input failure rate contains dropout rate column.
- Validate if input trial total follow-up (total duration) is a non-empty vector of positive integers.
- Validate if input simple is logical.

- Define a data frame with the start opening for enrollment at zero and cumulative duration. Add the event (or failure) time corresponding to the start of the enrollment. Finally, add the enrollment rate to the data frame corresponding to the start and end (failure) time. This will be recursively used to calculate the expected number of events later. For details, see vignette/eEventsTheory.Rmd
- Define a data frame including the cumulative duration of failure rates, the corresponding start time of the enrollment, failure rate and dropout rates. For details, see vignette/eEventsTheory.Rmd
- Only consider the failure rates in the interval of the end failure rate and total duration.
- Compute the failure rates over time using `stepfun` which is used to group rows by periods defined by fail_rate.
- Compute the dropout rate over time using `stepfun`.
- Compute the enrollment rate over time using `stepfun`. Details are available in vignette/eEventsTheory.Rmd.
- Compute expected events by interval at risk using the notations and descriptions in vignette/eEventsTheory.Rmd.
- Return `expected_event`

Examples

```
library(gsDesign2)

# Default arguments, simple output (total event count only)
expected_event()

# Event count by time period
expected_event(simple = FALSE)

# Early cutoff
expected_event(total_duration = .5)

# Single time period example
expected_event(
  enroll_rate = define_enroll_rate(duration = 10, rate = 10),
  fail_rate = define_fail_rate(duration = 100, fail_rate = log(2) / 6, dropout_rate = .01),
  total_duration = 22,
  simple = FALSE
)

# Single time period example, multiple enrollment periods
expected_event(
  enroll_rate = define_enroll_rate(duration = c(5, 5), rate = c(10, 20)),
  fail_rate = define_fail_rate(duration = 100, fail_rate = log(2) / 6, dropout_rate = .01),
  total_duration = 22, simple = FALSE
)

# Single time period example, multiple strata
enroll_rate <- define_enroll_rate(
  stratum = rep(c("Biomarker-positive", "Biomarker-negative"), each = 4),
  duration = c(2, 2, 2, 6, 2, 2, 2, 6),
  rate = c(1:4, 1:4))
failure_rate <- define_fail_rate(
```

```
  stratum = rep(c("Biomarker-positive", "Biomarker-negative"), each = 2),
  duration = c(3, 100, 3, 100),
  fail_rate = log(2) / c(8, 12, 8, 10),
  dropout_rate = 0.001)
# Number of expected events by stratum
sapply(c("Biomarker-positive", "Biomarker-negative"),
       function(ss){
         expected_event(enroll_rate = enroll_rate[enroll_rate$stratum == ss, ],
                        fail_rate = failure_rate[failure_rate$stratum == ss, ],
                        total_duration = 36,
                        simple = TRUE)})
```

`expected_time` *Predict time at which a targeted event count is achieved*

Description

`expected_time()` is made to match input format with `ahr()` and to solve for the time at which the expected accumulated events is equal to an input target. Enrollment and failure rate distributions are specified as follows. The piecewise exponential distribution allows a simple method to specify a distribution and enrollment pattern where the enrollment, failure and dropout rates changes over time.

Usage

```
expected_time(
  enroll_rate = define_enroll_rate(duration = c(2, 2, 10), rate = c(3, 6, 9) * 5),
  fail_rate = define_fail_rate(stratum = "All", duration = c(3, 100), fail_rate =
    log(2)/c(9, 18), hr = c(0.9, 0.6), dropout_rate = rep(0.001, 2)),
  target_event = 150,
  ratio = 1,
  interval = c(0.01, 100)
)
```

Arguments

`enroll_rate`	An `enroll_rate` data frame with or without stratum created by `define_enroll_rate()`.
`fail_rate`	A `fail_rate` data frame with or without stratum created by `define_fail_rate()`.
`target_event`	The targeted number of events to be achieved.
`ratio`	Experimental:Control randomization ratio.
`interval`	An interval that is presumed to include the time at which expected event count is equal to `target_event`.

Value

A data frame with `Time` (computed to match events in `target_event`), `AHR` (average hazard ratio), `Events` (`target_event` input), `info` (information under given scenarios), and `info0` (information under related null hypothesis) for each value of `total_duration` input.

Specification

- Use root-finding routine with ‘AHR()‘ to find time at which targeted events accrue.
- Return a data frame with a single row with the output from ‘AHR()‘ got the specified output.

Examples

```
# Example 1 ----
# default

expected_time()


# Example 2 ----
# check that result matches a finding using AHR()
# Start by deriving an expected event count
enroll_rate <- define_enroll_rate(duration = c(2, 2, 10), rate = c(3, 6, 9) * 5)
fail_rate <- define_fail_rate(
  duration = c(3, 100),
  fail_rate = log(2) / c(9, 18),
  hr = c(.9, .6),
  dropout_rate = .001
)
total_duration <- 20
xx <- ahr(enroll_rate, fail_rate, total_duration)
xx

# Next we check that the function confirms the timing of the final analysis.

expected_time(enroll_rate, fail_rate,
  target_event = xx$event, interval = c(.5, 1.5) * xx$time
)


# Example 3 ----
# In this example, we verify `expected_time()` by `ahr()`.

x <- ahr(
  enroll_rate = enroll_rate, fail_rate = fail_rate,
  ratio = 1, total_duration = 20
)

cat("The number of events by 20 months is ", x$event, ".\n")

y <- expected_time(
  enroll_rate = enroll_rate, fail_rate = fail_rate,
  ratio = 1, target_event = x$event
)

cat("The time to get ", x$event, " is ", y$time, "months.\n")
```

fixed_design_ahr — *Fixed design under non-proportional hazards*

Description

Computes fixed design sample size (given power) or power (given sample size) by:

- `fixed_design_ahr()` - Average hazard ratio method.
- `fixed_design_fh()` - Weighted logrank test with Fleming-Harrington weights (Farrington and Manning, 1990).
- `fixed_design_mb()` - Weighted logrank test with Magirr-Burman weights.
- `fixed_design_lf()` - Lachin-Foulkes method (Lachin and Foulkes, 1986).
- `fixed_design_maxcombo()` - MaxCombo method.
- `fixed_design_rmst()` - RMST method.
- `fixed_design_milestone()` - Milestone method.

Additionally, `fixed_design_rd()` provides fixed design for binary endpoint with treatment effect measuring in risk difference.

Usage

```
fixed_design_ahr(
  enroll_rate,
  fail_rate,
  alpha = 0.025,
  power = NULL,
  ratio = 1,
  study_duration = 36,
  event = NULL,
  info_scale = c("h0_h1_info", "h0_info", "h1_info")
)

fixed_design_fh(
  alpha = 0.025,
  power = NULL,
  ratio = 1,
  study_duration = 36,
  enroll_rate,
  fail_rate,
  rho = 0,
  gamma = 0,
  info_scale = c("h0_h1_info", "h0_info", "h1_info")
)

fixed_design_lf(
```

```
  alpha = 0.025,
  power = NULL,
  ratio = 1,
  study_duration = 36,
  enroll_rate,
  fail_rate
)

fixed_design_maxcombo(
  alpha = 0.025,
  power = NULL,
  ratio = 1,
  study_duration = 36,
  enroll_rate,
  fail_rate,
  rho = c(0, 0, 1),
  gamma = c(0, 1, 0),
  tau = rep(-1, 3)
)

fixed_design_mb(
  alpha = 0.025,
  power = NULL,
  ratio = 1,
  study_duration = 36,
  enroll_rate,
  fail_rate,
  tau = 6,
  w_max = Inf,
  info_scale = c("h0_h1_info", "h0_info", "h1_info")
)

fixed_design_milestone(
  alpha = 0.025,
  power = NULL,
  ratio = 1,
  enroll_rate,
  fail_rate,
  study_duration = 36,
  tau = NULL
)

fixed_design_rd(
  alpha = 0.025,
  power = NULL,
  ratio = 1,
  p_c,
  p_e,
```

```
  rd0 = 0,
  n = NULL,
  info_scale = c("h0_h1_info", "h0_info", "h1_info")
)

fixed_design_rmst(
  alpha = 0.025,
  power = NULL,
  ratio = 1,
  study_duration = 36,
  enroll_rate,
  fail_rate,
  tau = NULL
)
```

Arguments

enroll_rate	An enroll_rate data frame with or without stratum created by define_enroll_rate().
fail_rate	A fail_rate data frame with or without stratum created by define_fail_rate().
alpha	One-sided Type I error (strictly between 0 and 1).
power	Power (NULL to compute power or strictly between 0 and 1 - alpha otherwise).
ratio	Experimental:Control randomization ratio.
study_duration	Study duration.
event	A numerical vector specifying the targeted events at each analysis. See details.
info_scale	Information scale for calculation. Options are: • "h0_h1_info" (default): variance under both null and alternative hypotheses is used. • "h0_info": variance under null hypothesis is used. • "h1_info": variance under alternative hypothesis is used.
rho	A vector of numbers paring with gamma and tau for MaxCombo test.
gamma	A vector of numbers paring with rho and tau for MaxCombo test.
tau	Test parameter in RMST.
w_max	Test parameter of Magirr-Burman method.
p_c	A numerical value of the control arm rate.
p_e	A numerical value of the experimental arm rate.
rd0	Risk difference under null hypothesis, default is 0.
n	Sample size. If NULL with power input, the sample size will be computed to achieve the targeted power

Value

A list of design characteristic summary.

Examples

```
# AHR method ----

# Example 1: given power and compute sample size
x <- fixed_design_ahr(
  alpha = .025, power = .9,
  enroll_rate = define_enroll_rate(duration = 18, rate = 1),
  fail_rate = define_fail_rate(
    duration = c(4, 100),
    fail_rate = log(2) / 12,
    hr = c(1, .6),
    dropout_rate = .001
  ),
  study_duration = 36
)
x |> summary()

# Example 2: given sample size and compute power
x <- fixed_design_ahr(
  alpha = .025,
  enroll_rate = define_enroll_rate(duration = 18, rate = 20),
  fail_rate = define_fail_rate(
    duration = c(4, 100),
    fail_rate = log(2) / 12,
    hr = c(1, .6),
    dropout_rate = .001
  ),
  study_duration = 36
)
x |> summary()

# WLR test with FH weights ----

# Example 1: given power and compute sample size
x <- fixed_design_fh(
  alpha = .025, power = .9,
  enroll_rate = define_enroll_rate(duration = 18, rate = 1),
  fail_rate = define_fail_rate(
    duration = c(4, 100),
    fail_rate = log(2) / 12,
    hr = c(1, .6),
    dropout_rate = .001
  ),
  study_duration = 36,
  rho = 1, gamma = 1
)
x |> summary()

# Example 2: given sample size and compute power
x <- fixed_design_fh(
  alpha = .025,
  enroll_rate = define_enroll_rate(duration = 18, rate = 20),
```

```
  fail_rate = define_fail_rate(
    duration = c(4, 100),
    fail_rate = log(2) / 12,
    hr = c(1, .6),
    dropout_rate = .001
  ),
  study_duration = 36,
  rho = 1, gamma = 1
)
x |> summary()

# LF method ----

# Example 1: given power and compute sample size
x <- fixed_design_lf(
  alpha = .025, power = .9,
  enroll_rate = define_enroll_rate(duration = 18, rate = 1),
  fail_rate = define_fail_rate(
    duration = 100,
    fail_rate = log(2) / 12,
    hr = .7,
    dropout_rate = .001
  ),
  study_duration = 36
)
x |> summary()

# Example 2: given sample size and compute power
x <- fixed_design_lf(
  alpha = .025,
  enroll_rate = define_enroll_rate(duration = 18, rate = 20),
  fail_rate = define_fail_rate(
    duration = 100,
    fail_rate = log(2) / 12,
    hr = .7,
    dropout_rate = .001
  ),
  study_duration = 36
)
x |> summary()

# MaxCombo test ----

# Example 1: given power and compute sample size
x <- fixed_design_maxcombo(
  alpha = .025, power = .9,
  enroll_rate = define_enroll_rate(duration = 18, rate = 1),
  fail_rate = define_fail_rate(
    duration = c(4, 100),
    fail_rate = log(2) / 12,
    hr = c(1, .6),
    dropout_rate = .001
  ),
```

```
  study_duration = 36,
  rho = c(0, 0.5), gamma = c(0, 0), tau = c(-1, -1)
)
x |> summary()

# Example 2: given sample size and compute power
x <- fixed_design_maxcombo(
  alpha = .025,
  enroll_rate = define_enroll_rate(duration = 18, rate = 20),
  fail_rate = define_fail_rate(
    duration = c(4, 100),
    fail_rate = log(2) / 12,
    hr = c(1, .6),
    dropout_rate = .001
  ),
  study_duration = 36,
  rho = c(0, 0.5), gamma = c(0, 0), tau = c(-1, -1)
)
x |> summary()

# WLR test with MB weights ----

# Example 1: given power and compute sample size
x <- fixed_design_mb(
  alpha = .025, power = .9,
  enroll_rate = define_enroll_rate(duration = 18, rate = 1),
  fail_rate = define_fail_rate(
    duration = c(4, 100),
    fail_rate = log(2) / 12,
    hr = c(1, .6),
    dropout_rate = .001
  ),
  study_duration = 36,
  tau = 4,
  w_max = 2
)
x |> summary()

# Example 2: given sample size and compute power
x <- fixed_design_mb(
  alpha = .025,
  enroll_rate = define_enroll_rate(duration = 18, rate = 20),
  fail_rate = define_fail_rate(
    duration = c(4, 100),
    fail_rate = log(2) / 12,
    hr = c(1, .6),
    dropout_rate = .001
  ),
  study_duration = 36,
  tau = 4,
  w_max = 2
)
x |> summary()
```

```
# Milestone method ----

# Example 1: given power and compute sample size
x <- fixed_design_milestone(
  alpha = .025, power = .9,
  enroll_rate = define_enroll_rate(duration = 18, rate = 1),
  fail_rate = define_fail_rate(
    duration = 100,
    fail_rate = log(2) / 12,
    hr = .7,
    dropout_rate = .001
  ),
  study_duration = 36,
  tau = 18
)
x |> summary()

# Example 2: given sample size and compute power
x <- fixed_design_milestone(
  alpha = .025,
  enroll_rate = define_enroll_rate(duration = 18, rate = 20),
  fail_rate = define_fail_rate(
    duration = 100,
    fail_rate = log(2) / 12,
    hr = .7,
    dropout_rate = .001
  ),
  study_duration = 36,
  tau = 18
)
x |> summary()

# Binary endpoint with risk differences ----

# Example 1: given power and compute sample size
x <- fixed_design_rd(
  alpha = 0.025, power = 0.9, p_c = .15, p_e = .1,
  rd0 = 0, ratio = 1
)
x |> summary()

# Example 2: given sample size and compute power
x <- fixed_design_rd(
  alpha = 0.025, power = NULL, p_c = .15, p_e = .1,
  rd0 = 0, n = 2000, ratio = 1
)
x |> summary()

# RMST method ----

# Example 1: given power and compute sample size
x <- fixed_design_rmst(
```

```
    alpha = .025, power = .9,
    enroll_rate = define_enroll_rate(duration = 18, rate = 1),
    fail_rate = define_fail_rate(
      duration = 100,
      fail_rate = log(2) / 12,
      hr = .7,
      dropout_rate = .001
    ),
    study_duration = 36,
    tau = 18
  )
  x |> summary()

  # Example 2: given sample size and compute power
  x <- fixed_design_rmst(
    alpha = .025,
    enroll_rate = define_enroll_rate(duration = 18, rate = 20),
    fail_rate = define_fail_rate(
      duration = 100,
      fail_rate = log(2) / 12,
      hr = .7,
      dropout_rate = .001
    ),
    study_duration = 36,
    tau = 18
  )
  x |> summary()
```

---

gs_b *Default boundary generation*

Description

gs_b() is the simplest version of a function to be used with the upper and lower arguments in gs_power_npe() and gs_design_npe() or the upper_bound and lower_bound arguments in gs_prob_combo() and pmvnorm_combo(). It simply returns the vector of Z-values in the input vector par or, if k is specified, par[k] is returned. Note that if bounds need to change with changing information at analyses, gs_b() should not be used. For instance, for spending function bounds use gs_spending_bound().

Usage

```
gs_b(par = NULL, k = NULL, ...)
```

Arguments

par	For gs_b(), this is just Z-values for the boundaries; can include infinite values.
k	Is NULL (default), return par, else return par[k].
...	Further arguments passed to or from other methods.

Value

Returns the vector input `par` if `k` is `NULL`, otherwise, `par[k]`.

Specification

- Validate if the input k is null as default.
  - If the input k is null as default, return the whole vector of Z-values of the boundaries.
  - If the input k is not null, return the corresponding boundary in the vector of Z-values.
- Return a vector of boundaries.

Examples

```
# Simple: enter a vector of length 3 for bound
gs_b(par = 4:2)

# 2nd element of par
gs_b(par = 4:2, k = 2)

# Generate an efficacy bound using a spending function
# Use Lan-DeMets spending approximation of O'Brien-Fleming bound
# as 50%, 75% and 100% of final spending
# Information fraction
IF <- c(.5, .75, 1)
gs_b(par = gsDesign::gsDesign(
  alpha = .025, k = length(IF),
  test.type = 1, sfu = gsDesign::sfLDOF,
  timing = IF
)$upper$bound)
```

`gs_bound_summary` *Bound summary table*

Description

Summarizes the efficacy and futility bounds for each analysis.

Usage

```
gs_bound_summary(
  x,
  digits = 4,
  ddigits = 2,
  tdigits = 0,
  timename = "Month",
  alpha = NULL
)
```

Arguments

x	Design object.
digits	Number of digits past the decimal to be printed in the body of the table.
ddigits	Number of digits past the decimal to be printed for the natural parameter delta.
tdigits	Number of digits past the decimal point to be shown for estimated timing of each analysis.
timename	Text string indicating time unit.
alpha	Vector of alpha values to compute additional efficacy columns.

Value

A data frame

See Also

gsDesign::gsBoundSummary()

Examples

```
library(gsDesign2)

x <- gs_design_ahr(info_frac = c(.25, .75, 1), analysis_time = c(12, 25, 36))
gs_bound_summary(x)

x <- gs_design_wlr(info_frac = c(.25, .75, 1), analysis_time = c(12, 25, 36))
gs_bound_summary(x)

# Report multiple efficacy bounds (only supported for AHR designs)
x <- gs_design_ahr(analysis_time = 1:3*12, alpha = 0.0125)
gs_bound_summary(x, alpha = c(0.025, 0.05))
```

---

gs_cp — *Conditional power computation with non-constant effect size for non-/crossing an upper boundary at analysis j given observed Z value at analysis i*

Description

Conditional power computation with non-constant effect size for non-/crossing an upper boundary at analysis j given observed Z value at analysis i

Usage

```
gs_cp(x = NULL, theta = NULL, i = 1, zi = NULL)
```

Arguments

x	An object of type gsDesign2.
theta	Optional numeric vector with length $j - i + 1$, which specifies the natural parameter for treatment effect of interim analysis i through analysis j. The default is `NULL`.
i	Index of current analysis, with default of 1.
zi	Numeric scalar z-value observed at analysis i.

Details

We assume $Z_i, i = 1, ..., K$ are the z-statistics at an interim analysis i, respectively. We assume further $Z_i, i = 1, ..., K$ follows multivariate normal distribution

$$E(Z_i) = \theta_i \sqrt{I_i}$$

$$Cov(Z_i, Z_j) = I_i / I_j$$

. See https://merck.github.io/gsDesign2/articles/story-npe-background.html for assumption details.

The returned value is list of

$$P(\{Z_j \geq b_j\} \& \{\cap_{m=i+1}^{j-1} a_m \leq Z_m < b_m\} \mid Z_i = z_i).$$

$$P(\{Z_j \geq b_j\} \& \{\cap_{m=i+1}^{j-1} Z_m < b_m\} \mid Z_i = z_i).$$

$$P(\{Z_j \leq b_j\} \& \{\cap_{m=i+1}^{j-1} a_m \leq Z_m < b_m\} \mid Z_i = z_i).$$

Value

A list with the following elements:

`prob_alpha` A numeric vector of $(\alpha_{i,i+1}, \ldots, \alpha_{i,j-1}, \alpha_{i,j})$, where $\alpha_{i,j} = P(\{Z_j \geq b_j\} \& \{\cap_{m=i+1}^{j-1} a_m \leq Z_m < b_m\} \mid Z_i = z_i)$.

`prob_alpha_plus` A numeric vector of $(\alpha^+_{i,i+1}, \ldots, \alpha^+_{i,j-1}, \alpha^+_{i,j})$, where $\alpha^+_{i,j} = P(\{Z_j \geq b_j\} \& \{\cap_{m=i+1}^{j-1} Z_m < b_m\} \mid Z_i = z_i)$.

`prob_beta` A numeric vector of $(\beta_{i,i+1}, \ldots, \beta_{i,j-1}, \beta_{i,j})$, where $\beta_{i,j} = P(\{Z_j \leq b_j\} \& \{\cap_{m=i+1}^{j-1} a_m \leq Z_m < b_m\} \mid Z_i = z_i)$.

Examples

```
library(gsDesign2)
library(gsDesign)
library(dplyr)
library(mvtnorm)
# Example 1
# original design ----
enroll_rate <- define_enroll_rate(duration = c(2, 2, 2, 18),
                                  rate = c(1, 2, 3, 4))
fail_rate <- define_fail_rate(duration = c(3, Inf),
                              fail_rate = log(2) / 10,
                              dropout_rate = 0.001,
```

```
                              hr = c(1, 0.7))
x <- gs_design_ahr(enroll_rate = enroll_rate, fail_rate = fail_rate,
                   alpha = 0.025, beta = 0.1, ratio = 1,
                   info_frac = c(0.4, 0.6, 0.8, 1), analysis_time = 30,
                   binding = FALSE,
                   upper = gs_spending_bound,
                   upar = list(sf = sfLDOF, total_spend = 0.025, param = NULL),
                   lower = gs_spending_bound,
                   lpar = list(sf = sfLDOF, total_spend = 0.1),
                   h1_spending = TRUE,
                   test_lower = TRUE,
                   info_scale = "h0_h1_info") |> to_integer()

# calculate conditional power
# case 1: currently at IA1, compute conditional power at IA2, IA3 and FA,
# with default theta = NULL
gs_cp(x = x, i = 1,
zi = -gsDesign::hrn2z(hr = 0.8, n = 150+180, ratio = 1))

# case 2: currently at IA1, compute conditional power at IA2, IA3 and FA,
# with user-input theta
gs_cp(x = x,
  theta = c(0.15, 0.2, 0.25, 0.3),
  i = 1,
  zi = -gsDesign::hrn2z(hr = 0.8, n = 150+180, ratio = 1))
```

`gs_cp_simple` *Simple conditional power computation with non-constant effect size*

Description

Simple conditional power computation with non-constant effect size

Usage

```
gs_cp_simple(x = NULL, theta = NULL, i = 1, zi = NULL)
```

Arguments

x	An object of type gsDesign2.
theta	Optional numeric vector of natural parameter for treatment effects from analysis i through final analysis K. If `NULL`, values are taken from `x$analysis$theta`.
i	Integer index for current analysis. Default is 1.
zi	Numeric scalar z-value observed at analysis i.

Details

Suppose there are K analyses. Let Z_i and Z_j be the Z-statistics at a current analysis i and a future analysis j, respectively. Let's denote the statistical information at the i-th analysis as I_i. We further assume Z_i and Z_j are bivariate normal under standard group sequential assumptions with independent increments, then

$$E(Z_i) = \theta_i \sqrt{I_i}$$

$$Var(Z_i) = 1/I_i$$

$$Cov(Z_i, Z_j) = t \equiv I_i/I_j$$

See https://merck.github.io/gsDesign2/articles/story-npe-background.html for assumption details. Returned value is

$$P(Z_j > b_j \mid Z_i = z_i) = 1 - \Phi\left(\frac{b_j - \sqrt{t}z_i - \sqrt{I_j}(\theta_j - \theta_i\sqrt{t})}{\sqrt{1-t}}\right)$$

where b_j is the efficacy bound at analysis j, z_i is the observed Z-value at analysis i, and θ_i is the natural parameter for treatment effect at analysis i.

Value

A numeric vector with the conditional power $P(Z_j > b_j \mid Z_i = z_i)$ for $j = i+1, ..., K$.

Examples

```
library(gsDesign2)
library(gsDesign)
# Calculate conditional power with optional user-input theta
# (if NULL, will come from the input design)
alpha <- 0.025
beta <- 0.1
ratio <- 1
enroll_rate <- define_enroll_rate(
  duration = c(2, 2, 10),
  rate = (1:3) / 3)
fail_rate <- define_fail_rate(
  duration = Inf, fail_rate = log(2) / 9,
  hr = 0.6, dropout_rate = .0001)
analysis_time <- c(12, 24, 36)
ratio <- 1
upper <- gs_spending_bound
lower <- gs_b
upar <- list(sf = sfLDOF, total_spend = alpha)
lpar <- rep(-Inf, 3)
x <- gs_design_ahr(
  enroll_rate = enroll_rate, fail_rate = fail_rate,
  alpha = alpha, beta = beta, ratio = ratio,
  info_scale = "h0_h1_info",
  info_frac = NULL,
  analysis_time = c(12, 24, 36),
  upper = upper, upar = upar, test_upper = TRUE,
```

```
  lower = lower, lpar = lpar, test_lower = FALSE,
) |>
  to_integer()

# theta is NULL case
gs_cp_simple(x = x, theta = NULL, i = 1, zi = 1.5)

# User-input theta case
gs_cp_simple(x = x, theta = c(0.1, 0.2, 0.3), i = 1, zi = 1.5)
```

`gs_create_arm` *Create npsurvSS arm object*

Description

Create npsurvSS arm object

Usage

```
gs_create_arm(enroll_rate, fail_rate, ratio, total_time = 1e+06)
```

Arguments

`enroll_rate`	Enrollment rates from `define_enroll_rate()`.
`fail_rate`	Failure and dropout rates from `define_fail_rate()`.
`ratio`	Experimental:Control randomization ratio.
`total_time`	Total analysis time.

Value

A list of the two arms.

Specification

- Validate if there is only one stratum.
- Calculate the accrual duration.
- calculate the accrual intervals.
- Calculate the accrual parameter as the proportion of enrollment rate*duration.
- Set cure proportion to zero.
- set survival intervals and shape.
- Set fail rate in fail_rate to the Weibull scale parameter for the survival distribution in the arm 0.
- Set the multiplication of hazard ratio and fail rate to the Weibull scale parameter for the survival distribution in the arm 1.

- Set the shape parameter to one as the exponential distribution for shape parameter for the loss to follow-up distribution
- Set the scale parameter to one as the scale parameter for the loss to follow-up distribution since the exponential distribution is supported only
- Create arm 0 using `npsurvSS::create_arm()` using the parameters for arm 0.
- Create arm 1 using `npsurvSS::create_arm()` using the parameters for arm 1.
- Set the class of the two arms.
- Return a list of the two arms.

Examples

```
enroll_rate <- define_enroll_rate(
  duration = c(2, 2, 10),
  rate = c(3, 6, 9)
)

fail_rate <- define_fail_rate(
  duration = c(3, 100),
  fail_rate = log(2) / c(9, 18),
  hr = c(.9, .6),
  dropout_rate = .001
)

gs_create_arm(enroll_rate, fail_rate, ratio = 1)
```

`gs_design_ahr` *Calculate sample size and bounds given targeted power and Type I error in group sequential design using average hazard ratio under non-proportional hazards*

Description

Calculate sample size and bounds given targeted power and Type I error in group sequential design using average hazard ratio under non-proportional hazards

Usage

```
gs_design_ahr(
  enroll_rate = define_enroll_rate(duration = c(2, 2, 10), rate = c(3, 6, 9)),
  fail_rate = define_fail_rate(duration = c(3, 100), fail_rate = log(2)/c(9, 18), hr =
    c(0.9, 0.6), dropout_rate = 0.001),
  alpha = 0.025,
  beta = 0.1,
  info_frac = NULL,
  analysis_time = 36,
  ratio = 1,
  binding = FALSE,
```

```
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = alpha),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = beta),
  h1_spending = TRUE,
  test_upper = TRUE,
  test_lower = TRUE,
  info_scale = c("h0_h1_info", "h0_info", "h1_info"),
  r = 18,
  tol = 1e-06,
  interval = c(0.01, 1000)
)
```

Arguments

`enroll_rate` An `enroll_rate` data frame with or without stratum created by `define_enroll_rate()`.

`fail_rate` A `fail_rate` data frame with or without stratum created by `define_fail_rate()`.

`alpha` One-sided Type I error.

`beta` Type II error.

`info_frac` Targeted information fraction for analyses. See details.

`analysis_time` Targeted calendar timing of analyses. See details.

`ratio` Experimental:Control randomization ratio.

`binding` Indicator of whether futility bound is binding; default of `FALSE` is recommended.

`upper` Function to compute upper bound.

- `gs_spending_bound()`: alpha-spending efficacy bounds.
- `gs_b()`: fixed efficacy bounds.

`upar` Parameters passed to `upper`.

- If `upper` = `gs_b`, then `upar` is a numerical vector specifying the fixed efficacy bounds per analysis.
- If `upper` = `gs_spending_bound`, then `upar` is a list including
  - `sf` for the spending function family.
  - `total_spend` for total alpha spend.
  - `param` for the parameter of the spending function.
  - `timing` specifies spending time if different from information-based spending; see details.

`lower` Function to compute lower bound, which can be set up similarly as `upper`. See this vignette.

`lpar` Parameters passed to `lower`, which can be set up similarly as `upar`.

`h1_spending` Indicator that lower bound to be set by spending under alternate hypothesis (input `fail_rate`) if spending is used for lower bound. If this is `FALSE`, then the lower bound spending is under the null hypothesis. This is for two-sided symmetric or asymmetric testing under the null hypothesis; See this vignette.

test_upper Indicator of which analyses should include an upper (efficacy) bound; single value of TRUE (default) indicates all analyses; otherwise, a logical vector of the same length as info should indicate which analyses will have an efficacy bound.

test_lower Indicator of which analyses should include a lower bound; single value of TRUE (default) indicates all analyses; single value of FALSE indicated no lower bound; otherwise, a logical vector of the same length as info should indicate which analyses will have a lower bound.

info_scale Information scale for calculation. Options are:
- "h0_h1_info" (default): variance under both null and alternative hypotheses is used.
- "h0_info": variance under null hypothesis is used. This is often used for testing methods that use local alternatives, such as the Schoenfeld method.
- "h1_info": variance under alternative hypothesis is used.

r Integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally, r will not be changed by the user.

tol Tolerance parameter for boundary convergence (on Z-scale); normally not changed by the user.

interval An interval presumed to include the times at which expected event count is equal to targeted event. Normally, this can be ignored by the user as it is set to c(.01, 1000).

Details

The parameters info_frac and analysis_time are used to determine the timing for interim and final analyses.

- If the interim analysis is determined by targeted information fraction and the study duration is known, then info_frac is a numerical vector where each element (greater than 0 and less than or equal to 1) represents the information fraction for each analysis. The analysis_time, which defaults to 36, indicates the time for the final analysis.
- If interim analyses are determined solely by the targeted calendar analysis timing from start of study, then analysis_time will be a vector specifying the time for each analysis.
- If both the targeted analysis time and the targeted information fraction are utilized for a given analysis, then timing will be the maximum of the two with both info_frac and analysis_time provided as vectors.

Value

A list with input parameters, enrollment rate, analysis, and bound.

- The $input is a list including alpha, beta, ratio, etc.
- The $enroll_rate is a table showing the enrollment for arriving the targeted power (1 - beta).
- The $fail_rate is a table showing the failure and dropout rates, which is the same as input.
- The $bound is a table summarizing the efficacy and futility bound per analysis.
- The analysis is a table summarizing the analysis time, sample size, events, average HR, treatment effect and statistical information per analysis.

Specification

- Validate if input analysis_time is a positive number or positive increasing sequence.
- Validate if input info_frac is a positive number or positive increasing sequence on (0, 1] with final value of 1.
- Validate if input info_frac and analysis_time have the same length if both have length > 1.
- Get information at input analysis_time
  - Use `gs_info_ahr()` to get the information and effect size based on AHR approximation.
  - Extract the final event.
  - Check if input If needed for (any) interim analysis timing.
- Add the analysis column to the information at input analysis_time.
- Add the sample size column to the information at input analysis_time using `expected_accural()`.
- Get sample size and bounds using `gs_design_npe()` and save them to bounds.
- Add Time, Events, AHR, N that have already been calculated to the bounds.
- Return a list of design enrollment, failure rates, and bounds.

Examples

```
library(gsDesign)
library(gsDesign2)

# Example 1 ----
# call with defaults
gs_design_ahr()

# Example 2 ----
# Single analysis
gs_design_ahr(analysis_time = 40)

# Example 3 ----
# Multiple analysis_time
gs_design_ahr(analysis_time = c(12, 24, 36))

# Example 4 ----
# Specified information fraction

gs_design_ahr(info_frac = c(.25, .75, 1), analysis_time = 36)


# Example 5 ----
# multiple analysis times & info_frac
# driven by times
gs_design_ahr(info_frac = c(.25, .75, 1), analysis_time = c(12, 25, 36))
# driven by info_frac

gs_design_ahr(info_frac = c(1 / 3, .8, 1), analysis_time = c(12, 25, 36))


# Example 6 ----
```

```
# 2-sided symmetric design with O'Brien-Fleming spending

gs_design_ahr(
  analysis_time = c(12, 24, 36),
  binding = TRUE,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  h1_spending = FALSE
)

# 2-sided asymmetric design with O'Brien-Fleming upper spending
# Pocock lower spending under H1 (NPH)

gs_design_ahr(
  analysis_time = c(12, 24, 36),
  binding = TRUE,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDPocock, total_spend = 0.1, param = NULL, timing = NULL),
  h1_spending = TRUE
)


# Example 7 ----

gs_design_ahr(
  alpha = 0.0125,
  analysis_time = c(12, 24, 36),
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.0125, param = NULL, timing = NULL),
  lower = gs_b,
  lpar = rep(-Inf, 3)
)

gs_design_ahr(
  alpha = 0.0125,
  analysis_time = c(12, 24, 36),
  upper = gs_b,
  upar = gsDesign::gsDesign(
    k = 3, test.type = 1, n.I = c(.25, .75, 1),
    sfu = sfLDOF, sfupar = NULL, alpha = 0.0125
  )$upper$bound,
  lower = gs_b,
  lpar = rep(-Inf, 3)
)
```

gs_design_combo *Group sequential design using MaxCombo test under non-proportional hazards*

Description

Group sequential design using MaxCombo test under non-proportional hazards

Usage

```
gs_design_combo(
  enroll_rate = define_enroll_rate(duration = 12, rate = 500/12),
 fail_rate = define_fail_rate(duration = c(4, 100), fail_rate = log(2)/15, hr = c(1,
    0.6), dropout_rate = 0.001),
 fh_test = rbind(data.frame(rho = 0, gamma = 0, tau = -1, test = 1, analysis = 1:3,
   analysis_time = c(12, 24, 36)), data.frame(rho = c(0, 0.5), gamma = 0.5, tau = -1,
    test = 2:3, analysis = 3, analysis_time = 36)),
  ratio = 1,
  alpha = 0.025,
  beta = 0.2,
  binding = FALSE,
  upper = gs_b,
  upar = c(3, 2, 1),
  lower = gs_b,
  lpar = c(-1, 0, 1),
  algorithm = mvtnorm::GenzBretz(maxpts = 1e+05, abseps = 1e-05),
  n_upper_bound = 1000,
  ...
)
```

Arguments

enroll_rate	An enroll_rate data frame with or without stratum created by define_enroll_rate().
fail_rate	A fail_rate data frame with or without stratum created by define_fail_rate().
fh_test	A data frame to summarize the test in each analysis. See examples for its data structure.
ratio	Experimental:Control randomization ratio.
alpha	One-sided Type I error.
beta	Type II error.
binding	Indicator of whether futility bound is binding; default of FALSE is recommended.
upper	Function to compute upper bound. • gs_spending_bound(): alpha-spending efficacy bounds. • gs_b(): fixed efficacy bounds.
upar	Parameters passed to upper. • If upper = gs_b, then upar is a numerical vector specifying the fixed efficacy bounds per analysis.

- If upper = gs_spending_bound, then upar is a list including
  - sf for the spending function family.
  - total_spend for total alpha spend.
  - param for the parameter of the spending function.
  - timing specifies spending time if different from information-based spending; see details.

lower	Function to compute lower bound, which can be set up similarly as upper. See this vignette.
lpar	Parameters passed to lower, which can be set up similarly as upar.
algorithm	an object of class GenzBretz, Miwa or TVPACK specifying both the algorithm to be used as well as the associated hyper parameters.
n_upper_bound	A numeric value of upper limit of sample size.
...	Additional parameters passed to mvtnorm::pmvnorm.

Value

A list with input parameters, enrollment rate, analysis, and bound.

Examples

```
# The example is slow to run
library(mvtnorm)
library(gsDesign)

enroll_rate <- define_enroll_rate(
  duration = 12,
  rate = 500 / 12
)

fail_rate <- define_fail_rate(
  duration = c(4, 100),
  fail_rate = log(2) / 15, # median survival 15 month
  hr = c(1, .6),
  dropout_rate = 0.001
)

fh_test <- rbind(
  data.frame(
    rho = 0, gamma = 0, tau = -1,
    test = 1, analysis = 1:3, analysis_time = c(12, 24, 36)
  ),
  data.frame(
    rho = c(0, 0.5), gamma = 0.5, tau = -1,
    test = 2:3, analysis = 3, analysis_time = 36
  )
)

x <- gsSurv(
  k = 3,
```

```
  test.type = 4,
  alpha = 0.025,
  beta = 0.2,
  astar = 0,
  timing = 1,
  sfu = sfLDOF,
  sfupar = 0,
  sfl = sfLDOF,
  sflpar = 0,
  lambdaC = 0.1,
  hr = 0.6,
  hr0 = 1,
  eta = 0.01,
  gamma = 10,
  R = 12,
  S = NULL,
  T = 36,
  minfup = 24,
  ratio = 1
)

# Example 1 ----
# User-defined boundary

gs_design_combo(
  enroll_rate,
  fail_rate,
  fh_test,
  alpha = 0.025, beta = 0.2,
  ratio = 1,
  binding = FALSE,
  upar = x$upper$bound,
  lpar = x$lower$bound
)

# Example 2 ----

# Boundary derived by spending function
gs_design_combo(
  enroll_rate,
  fail_rate,
  fh_test,
  alpha = 0.025,
  beta = 0.2,
  ratio = 1,
  binding = FALSE,
  upper = gs_spending_combo,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025), # alpha spending
  lower = gs_spending_combo,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.2), # beta spending
)
```

gs_design_npe — *Group sequential design computation with non-constant effect and information.*

Description

The following two functions allow a non-constant treatment effect over time, but also can be applied for the usual homogeneous effect size designs. They require treatment effect and statistical information at each analysis as well as a method of deriving bounds, such as spending. Initial bound types supported are spending bounds and fixed bounds. These routines enables two things not available in the gsDesign package: 1) non-constant effect, 2) more flexibility in boundary selection.

`gs_power_npe()` derives group sequential bounds and boundary crossing probabilities for a design, given treatment effect and information at each analysis and the method of deriving bounds, such as spending.

`gs_design_npe()` derives group sequential design size, bounds and boundary crossing probabilities based on proportionate information and effect size at analyses, as well as the method of deriving bounds, such as spending.

The only differences in arguments between the two functions are the `alpha` and `beta` parameters used in the `gs_design_npe()`.

Usage

```
gs_design_npe(
  theta = 0.1,
  theta0 = 0,
  theta1 = theta,
  info = 1,
  info0 = NULL,
  info1 = NULL,
  info_scale = c("h0_h1_info", "h0_info", "h1_info"),
  alpha = 0.025,
  beta = 0.1,
  upper = gs_b,
  upar = qnorm(0.975),
  lower = gs_b,
  lpar = -Inf,
  test_upper = TRUE,
  test_lower = TRUE,
  binding = FALSE,
  r = 18,
  tol = 1e-06
)

gs_power_npe(
  theta = 0.1,
  theta0 = 0,
```

```
  theta1 = theta,
  info = 1,
  info0 = NULL,
  info1 = NULL,
  info_scale = c("h0_h1_info", "h0_info", "h1_info"),
  upper = gs_b,
  upar = qnorm(0.975),
  lower = gs_b,
  lpar = -Inf,
  test_upper = TRUE,
  test_lower = TRUE,
  binding = FALSE,
  r = 18,
  tol = 1e-06
)
```

Arguments

`theta` Natural parameter for group sequential design representing expected cumulative drift at all analyses; used for power calculation. It can be a scalar (constant treatment effect) or a vector (non-constant treatment effect). The user must provide a value for `theta`.

`theta0` Natural parameter for null hypothesis. It can be a scalar (constant treatment effect) or a vector (non-constant treatment effect). The default is 0. If a value other than 0 is provided, it affects upper bound computation.

`theta1` Natural parameter for alternate hypothesis, if needed for lower bound computation. It can be a scalar (constant treatment effect) or a vector (non-constant treatment effect). The default is the same as `theta`, which yields the usual beta-spending. If set to 0, spending is 2-sided under the null hypothesis.

`info` Statistical information at all analyses for input `theta`. It is a vector of positive numbers in increasing order. The user must provide values corresponding to `theta`.

`info0` Statistical information under null hypothesis. It is a vector of all positive numbers with increasing order. Default is set to be the same as `info`. If `info0` is different than `info`, it impacts null hypothesis bound calculation.

`info1` Statistical information under hypothesis used for futility bound calculation. It is a vector of all positive numbers with increasing order. Default is set to be the same as `info`. If `info1` is different from `info`, it impacts futility bound calculation.

`info_scale` Information scale for calculation. Options are:

- `"h0_h1_info"` (default): variance under both null and alternative hypotheses is used.
- `"h0_info"`: variance under null hypothesis is used. This is often used for testing methods that use local alternatives, such as the Schoenfeld method.
- `"h1_info"`: variance under alternative hypothesis is used.

`alpha` One-sided Type I error.

`beta`	Type II error.
`upper`	Function to compute upper bound. • `gs_spending_bound()`: alpha-spending efficacy bounds. • `gs_b()`: fixed efficacy bounds.
`upar`	Parameters passed to `upper`. • If `upper = gs_b`, then `upar` is a numerical vector specifying the fixed efficacy bounds per analysis. • If `upper = gs_spending_bound`, then `upar` is a list including – `sf` for the spending function family. – `total_spend` for total alpha spend. – `param` for the parameter of the spending function. – `timing` specifies spending time if different from information-based spending; see details.
`lower`	Function to compute lower bound, which can be set up similarly as `upper`. See this vignette.
`lpar`	Parameters passed to `lower`, which can be set up similarly as `upar`.
`test_upper`	Indicator of which analyses should include an upper (efficacy) bound; single value of `TRUE` (default) indicates all analyses; otherwise, a logical vector of the same length as `info` should indicate which analyses will have an efficacy bound.
`test_lower`	Indicator of which analyses should include a lower bound; single value of `TRUE` (default) indicates all analyses; single value of `FALSE` indicated no lower bound; otherwise, a logical vector of the same length as `info` should indicate which analyses will have a lower bound.
`binding`	Indicator of whether futility bound is binding; default of `FALSE` is recommended.
`r`	Integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally, `r` will not be changed by the user.
`tol`	Tolerance parameter for boundary convergence (on Z-scale); normally not changed by the user.

Details

The bound specifications (`upper`, `lower`, `upar`, `lpar`) of `gs_design_npe()` will be used to ensure Type I error and other boundary properties are as specified. See the help file of `gs_spending_bound()` for details on spending function.

Value

A tibble with columns of

- `analysis`: analysis index.
- `bound`: either of value `"upper"` or `"lower"`, indicating the upper and lower bound.
- `z`: the Z-score bounds.
- `probability`: cumulative probability of crossing the bound at or before the analysis.

- theta: same as the input.
- theta1: same as the input.
- info: statistical information at each analysis.
  - If it is returned by gs_power_npe, the info, info0, info1 are same as the input.
  - If it is returned by gs_design_npe, the info, info0, info1 are changed by a constant scale factor. factor to ensure the design has power 1 - beta.
- info0: statistical information under the null at each analysis.
- info1: statistical information under the alternative at each analysis.
- info_frac: information fraction at each analysis, i.e., info / max(info).

Specification

- Validate if input info is a numeric vector or NULL, if non-NULL validate if it is strictly increasing and positive.
- Validate if input info0 is a numeric vector or NULL, if non-NULL validate if it is strictly increasing and positive.
- Validate if input info1 is a numeric vector or NULL, if non-NULL validate if it is strictly increasing and positive.
- Validate if input theta is a real vector and has the same length as info.
- Validate if input theta1 is a real vector and has the same length as info.
- Validate if input test_upper and test_lower are logical and have the same length as info.
- Validate if input test_upper value is TRUE.
- Validate if input alpha and beta are positive and of length one.
- Validate if input alpha and beta are from the unit interval and alpha is smaller than beta.
- Initialize bounds, numerical integration grids, boundary crossing probabilities.
- Compute fixed sample size for desired power and Type I error.
- Find an interval for information inflation to give correct power using gs_power_npe().
-
- If there is no interim analysis, return a tibble including Analysis time, upper bound, Z-value, Probability of crossing bound, theta, info0 and info1.
- If the design is a group sequential design, return a tibble of Analysis, Bound, Z, Probability, theta, info, info0.

- Extract the length of input info as the number of interim analysis.
- Validate if input info0 is NULL, so set it equal to info.
- Validate if the length of inputs info and info0 are the same.
- Validate if input theta is a scalar, so replicate the value for all k interim analysis.
- Validate if input theta1 is NULL and if it is a scalar. If it is NULL, set it equal to input theta. If it is a scalar, replicate the value for all k interim analysis.
- Validate if input test_upper is a scalar, so replicate the value for all k interim analysis.
- Validate if input test_lower is a scalar, so replicate the value for all k interim analysis.

- Define vector a to be -Inf with length equal to the number of interim analysis.
- Define vector b to be Inf with length equal to the number of interim analysis.
- Define hgm1_0 and hgm1 to be NULL.
- Define upper_prob and lower_prob to be vectors of NA with length of the number of interim analysis.
- Update lower and upper bounds using `gs_b()`.
- If there are no interim analysis, compute probabilities of crossing upper and lower bounds using `h1()`.
- Compute cross upper and lower bound probabilities using `hupdate()` and `h1()`.
- Return a tibble of analysis number, bound, z-values, probability of crossing bounds, theta, theta1, info, and info0.

Author(s)

Keaven Anderson <keaven_anderson@merck.com>

Examples

```
library(gsDesign)

# Example 1 ----
# gs_design_npe with single analysis
# Lachin book p 71 difference of proportions example
pc <- .28 # Control response rate
pe <- .40 # Experimental response rate
p0 <- (pc + pe) / 2 # Ave response rate under H0

# Information per increment of 1 in sample size
info0 <- 1 / (p0 * (1 - p0) * 4)
info <- 1 / (pc * (1 - pc) * 2 + pe * (1 - pe) * 2)

# Result should round up to next even number = 652
# Divide information needed under H1 by information per patient added
gs_design_npe(theta = pe - pc, info = info, info0 = info0)


# Example 2 ----
# gs_design_npe with with fixed bound
x <- gs_design_npe(
  alpha = 0.0125,
  theta = c(.1, .2, .3),
  info = (1:3) * 80,
  info0 = (1:3) * 80,
  upper = gs_b,
  upar = gsDesign::gsDesign(k = 3, sfu = gsDesign::sfLDOF, alpha = 0.0125)$upper$bound,
  lower = gs_b,
  lpar = c(-1, 0, 0)
)
x
```

```
# Same upper bound; this represents non-binding Type I error and will total 0.025
gs_power_npe(
  theta = rep(0, 3),
  info = (x |> dplyr::filter(bound == "upper"))$info,
  upper = gs_b,
  upar = (x |> dplyr::filter(bound == "upper"))$z,
  lower = gs_b,
  lpar = rep(-Inf, 3)
)

# Example 3 ----
# gs_design_npe with spending bound
# Design with futility only at analysis 1; efficacy only at analyses 2, 3
# Spending bound for efficacy; fixed bound for futility
# NOTE: test_upper and test_lower DO NOT WORK with gs_b; must explicitly make bounds infinite
# test_upper and test_lower DO WORK with gs_spending_bound
gs_design_npe(
  theta = c(.1, .2, .3),
  info = (1:3) * 40,
  info0 = (1:3) * 40,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lower = gs_b,
  lpar = c(-1, -Inf, -Inf),
  test_upper = c(FALSE, TRUE, TRUE)
)

# one can try `info_scale = "h1_info"` or `info_scale = "h0_info"` here
gs_design_npe(
  theta = c(.1, .2, .3),
  info = (1:3) * 40,
  info0 = (1:3) * 30,
  info_scale = "h1_info",
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lower = gs_b,
  lpar = c(-1, -Inf, -Inf),
  test_upper = c(FALSE, TRUE, TRUE)
)

# Example 4 ----
# gs_design_npe with spending function bounds
# 2-sided asymmetric bounds
# Lower spending based on non-zero effect
gs_design_npe(
  theta = c(.1, .2, .3),
  info = (1:3) * 40,
  info0 = (1:3) * 30,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfHSD, total_spend = 0.1, param = -1, timing = NULL)
```

```
)

# Example 5 ----
# gs_design_npe with two-sided symmetric spend, O'Brien-Fleming spending
# Typically, 2-sided bounds are binding
xx <- gs_design_npe(
  theta = c(.1, .2, .3),
  info = (1:3) * 40,
  binding = TRUE,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL)
)
xx

# Re-use these bounds under alternate hypothesis
# Always use binding = TRUE for power calculations
gs_power_npe(
  theta = c(.1, .2, .3),
  info = (1:3) * 40,
  binding = TRUE,
  upper = gs_b,
  lower = gs_b,
  upar = (xx |> dplyr::filter(bound == "upper"))$z,
  lpar = -(xx |> dplyr::filter(bound == "upper"))$z
)

# Example 6 ----
# Default of gs_power_npe (single analysis; Type I error controlled)
gs_power_npe(theta = 0) |> dplyr::filter(bound == "upper")

# Example 7 ----
# gs_power_npe with fixed bound
gs_power_npe(
  theta = c(.1, .2, .3),
  info = (1:3) * 40,
  upper = gs_b,
  upar = gsDesign::gsDesign(k = 3, sfu = gsDesign::sfLDOF)$upper$bound,
  lower = gs_b,
  lpar = c(-1, 0, 0)
)

# Same fixed efficacy bounds, no futility bound (i.e., non-binding bound), null hypothesis
gs_power_npe(
  theta = rep(0, 3),
  info = (1:3) * 40,
  upar = gsDesign::gsDesign(k = 3, sfu = gsDesign::sfLDOF)$upper$bound,
  lpar = rep(-Inf, 3)
) |>
  dplyr::filter(bound == "upper")

# Example 8 ----
```

```
# gs_power_npe with fixed bound testing futility only at analysis 1; efficacy only at analyses 2, 3
gs_power_npe(
  theta = c(.1, .2, .3),
  info = (1:3) * 40,
  upper = gs_b,
  upar = c(Inf, 3, 2),
  lower = gs_b,
  lpar = c(qnorm(.1), -Inf, -Inf)
)

# Example 9 ----
# gs_power_npe with spending function bounds
# Lower spending based on non-zero effect
gs_power_npe(
  theta = c(.1, .2, .3),
  info = (1:3) * 40,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfHSD, total_spend = 0.1, param = -1, timing = NULL)
)

# Same bounds, but power under different theta
gs_power_npe(
  theta = c(.15, .25, .35),
  info = (1:3) * 40,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfHSD, total_spend = 0.1, param = -1, timing = NULL)
)

# Example 10 ----
# gs_power_npe with two-sided symmetric spend, O'Brien-Fleming spending
# Typically, 2-sided bounds are binding
x <- gs_power_npe(
  theta = rep(0, 3),
  info = (1:3) * 40,
  binding = TRUE,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL)
)

# Re-use these bounds under alternate hypothesis
# Always use binding = TRUE for power calculations
gs_power_npe(
  theta = c(.1, .2, .3),
  info = (1:3) * 40,
  binding = TRUE,
  upar = (x |> dplyr::filter(bound == "upper"))$z,
  lpar = -(x |> dplyr::filter(bound == "upper"))$z
```

```
)

# Example 11 ----
# Different values of `r` and `tol` lead to different numerical accuracy
# Larger `r` and smaller `tol` give better accuracy, but leads to slow computation
n_analysis <- 5
gs_power_npe(
  theta = 0.1,
  info = 1:n_analysis,
  info0 = 1:n_analysis,
  info1 = NULL,
  info_scale = "h0_info",
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lower = gs_b,
  lpar = -rep(Inf, n_analysis),
  test_upper = TRUE,
  test_lower = FALSE,
  binding = FALSE,
  # Try different combinations of (r, tol) with
  # r in 6, 18, 24, 30, 35, 40, 50, 60, 70, 80, 90, 100
  # tol in 1e-6, 1e-12
  r = 6,
  tol = 1e-6
)
```

---

gs_design_rd *Group sequential design of binary outcome measuring in risk difference*

Description

Group sequential design of binary outcome measuring in risk difference

Usage

```
gs_design_rd(
  p_c = tibble::tibble(stratum = "All", rate = 0.2),
  p_e = tibble::tibble(stratum = "All", rate = 0.15),
  info_frac = 1:3/3,
  rd0 = 0,
  alpha = 0.025,
  beta = 0.1,
  ratio = 1,
  stratum_prev = NULL,
  weight = c("unstratified", "ss", "invar", "mr"),
  upper = gs_b,
  lower = gs_b,
  upar = gsDesign(k = 3, test.type = 1, sfu = sfLDOF, sfupar = NULL)$upper$bound,
```

```
  lpar = c(qnorm(0.1), rep(-Inf, 2)),
  test_upper = TRUE,
  test_lower = TRUE,
  info_scale = c("h0_h1_info", "h0_info", "h1_info"),
  binding = FALSE,
  r = 18,
  tol = 1e-06,
  h1_spending = TRUE
)
```

Arguments

p_c	Rate at the control group.
p_e	Rate at the experimental group.
info_frac	Statistical information fraction.
rd0	Treatment effect under super-superiority designs, the default is 0.
alpha	One-sided Type I error.
beta	Type II error.
ratio	Experimental:Control randomization ratio (not yet implemented).
stratum_prev	Randomization ratio of different stratum. If it is unstratified design then `NULL`. Otherwise it is a tibble containing two columns (stratum and prevalence).
weight	The weighting scheme for stratified population.
upper	Function to compute upper bound.
lower	Function to compute lower bound.
upar	Parameters passed to `upper`.
lpar	Parameters passed to `lower`.
test_upper	Indicator of which analyses should include an upper (efficacy) bound; single value of `TRUE` (default) indicates all analyses; otherwise, a logical vector of the same length as `info` should indicate which analyses will have an efficacy bound.
test_lower	Indicator of which analyses should include an lower bound; single value of `TRUE` (default) indicates all analyses; single value of `FALSE` indicates no lower bound; otherwise, a logical vector of the same length as `info` should indicate which analyses will have a lower bound.
info_scale	Information scale for calculation. Options are: • `"h0_h1_info"` (default): variance under both null and alternative hypotheses is used. • `"h0_info"`: variance under null hypothesis is used. • `"h1_info"`: variance under alternative hypothesis is used.
binding	Indicator of whether futility bound is binding; default of `FALSE` is recommended.
r	Integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally, `r` will not be changed by the user.
tol	Tolerance parameter for boundary convergence (on Z-scale).
h1_spending	Indicator that lower bound to be set by spending under alternate hypothesis (input `fail_rate`) if spending is used for lower bound.

Details

To be added.

Value

A list with input parameters, analysis, and bound.

Examples

```
library(gsDesign)

# Example 1 ----
# unstratified group sequential design
x <- gs_design_rd(
  p_c = tibble::tibble(stratum = "All", rate = .2),
  p_e = tibble::tibble(stratum = "All", rate = .15),
  info_frac = c(0.7, 1),
  rd0 = 0,
  alpha = .025,
  beta = .1,
  ratio = 1,
  stratum_prev = NULL,
  weight = "unstratified",
  upper = gs_b,
  lower = gs_b,
  upar = gsDesign(k = 2, test.type = 1, sfu = sfLDOF, sfupar = NULL)$upper$bound,
  lpar = c(qnorm(.1), rep(-Inf, 2))
)

y <- gs_power_rd(
  p_c = tibble::tibble(stratum = "All", rate = .2),
  p_e = tibble::tibble(stratum = "All", rate = .15),
  n = tibble::tibble(stratum = "All", n = x$analysis$n, analysis = 1:2),
  rd0 = 0,
  ratio = 1,
  weight = "unstratified",
  upper = gs_b,
  lower = gs_b,
  upar = gsDesign(k = 2, test.type = 1, sfu = sfLDOF, sfupar = NULL)$upper$bound,
  lpar = c(qnorm(.1), rep(-Inf, 2))
)

# The above 2 design share the same power with the same sample size and treatment effect
x$bound$probability[x$bound$bound == "upper" & x$bound$analysis == 2]
y$bound$probability[y$bound$bound == "upper" & y$bound$analysis == 2]

# Example 2 ----
# stratified group sequential design
gs_design_rd(
  p_c = tibble::tibble(
    stratum = c("biomarker positive", "biomarker negative"),
    rate = c(.2, .25)
```

```
  ),
  p_e = tibble::tibble(
    stratum = c("biomarker positive", "biomarker negative"),
    rate = c(.15, .22)
  ),
  info_frac = c(0.7, 1),
  rd0 = 0,
  alpha = .025,
  beta = .1,
  ratio = 1,
  stratum_prev = tibble::tibble(
    stratum = c("biomarker positive", "biomarker negative"),
    prevalence = c(.4, .6)
  ),
  weight = "ss",
  upper = gs_spending_bound, lower = gs_b,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lpar = rep(-Inf, 2)
)
```

---

gs_design_wlr *Group sequential design using weighted log-rank test under non-proportional hazards*

Description

Group sequential design using weighted log-rank test under non-proportional hazards

Usage

```
gs_design_wlr(
  enroll_rate = define_enroll_rate(duration = c(2, 2, 10), rate = c(3, 6, 9)),
 fail_rate = tibble(stratum = "All", duration = c(3, 100), fail_rate = log(2)/c(9, 18),
    hr = c(0.9, 0.6), dropout_rate = rep(0.001, 2)),
  weight = "logrank",
  approx = "asymptotic",
  alpha = 0.025,
  beta = 0.1,
  ratio = 1,
  info_frac = NULL,
  info_scale = c("h0_h1_info", "h0_info", "h1_info"),
  analysis_time = 36,
  binding = FALSE,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = alpha),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = beta),
  test_upper = TRUE,
```

```
  test_lower = TRUE,
  h1_spending = TRUE,
  r = 18,
  tol = 1e-06,
  interval = c(0.01, 1000)
)
```

Arguments

`enroll_rate` An `enroll_rate` data frame with or without stratum created by `define_enroll_rate()`.

`fail_rate` A `fail_rate` data frame with or without stratum created by `define_fail_rate()`.

`weight` Weight of weighted log rank test:

- `"logrank"` = regular logrank test.
- `list(method = "fh", param = list(rho = ..., gamma = ...))` = Fleming-Harrington weighting functions.
- `list(method = "mb", param = list(tau = ..., w_max = ...))` = Magirr and Burman weighting functions.

`approx` Approximate estimation method for Z statistics.

- `"event_driven"` = only work under proportional hazard model with log rank test.
- `"asymptotic"`.

`alpha` One-sided Type I error.

`beta` Type II error.

`ratio` Experimental:Control randomization ratio.

`info_frac` Targeted information fraction for analyses. See details.

`info_scale` Information scale for calculation. Options are:

- `"h0_h1_info"` (default): variance under both null and alternative hypotheses is used.
- `"h0_info"`: variance under null hypothesis is used. This is often used for testing methods that use local alternatives, such as the Schoenfeld method.
- `"h1_info"`: variance under alternative hypothesis is used.

`analysis_time` Targeted calendar timing of analyses. See details.

`binding` Indicator of whether futility bound is binding; default of `FALSE` is recommended.

`upper` Function to compute upper bound.

- `gs_spending_bound()`: alpha-spending efficacy bounds.
- `gs_b()`: fixed efficacy bounds.

`upar` Parameters passed to `upper`.

- If `upper = gs_b`, then `upar` is a numerical vector specifying the fixed efficacy bounds per analysis.
- If `upper = gs_spending_bound`, then `upar` is a list including
  - `sf` for the spending function family.
  - `total_spend` for total alpha spend.

- param for the parameter of the spending function.
- timing specifies spending time if different from information-based spending; see details.

lower
: Function to compute lower bound, which can be set up similarly as upper. See this vignette.

lpar
: Parameters passed to lower, which can be set up similarly as upar.

test_upper
: Indicator of which analyses should include an upper (efficacy) bound; single value of TRUE (default) indicates all analyses; otherwise, a logical vector of the same length as info should indicate which analyses will have an efficacy bound.

test_lower
: Indicator of which analyses should include a lower bound; single value of TRUE (default) indicates all analyses; single value of FALSE indicated no lower bound; otherwise, a logical vector of the same length as info should indicate which analyses will have a lower bound.

h1_spending
: Indicator that lower bound to be set by spending under alternate hypothesis (input fail_rate) if spending is used for lower bound. If this is FALSE, then the lower bound spending is under the null hypothesis. This is for two-sided symmetric or asymmetric testing under the null hypothesis; See this vignette.

r
: Integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally, r will not be changed by the user.

tol
: Tolerance parameter for boundary convergence (on Z-scale); normally not changed by the user.

interval
: An interval presumed to include the times at which expected event count is equal to targeted event. Normally, this can be ignored by the user as it is set to c(.01, 1000).

Value

A list with input parameters, enrollment rate, analysis, and bound.

Specification

- Validate if input analysis_time is a positive number or a positive increasing sequence.
- Validate if input info_frac is a positive number or positive increasing sequence on (0, 1] with final value of 1.
- Validate if inputs info_frac and analysis_time have the same length if both have length > 1.
- Compute information at input analysis_time using gs_info_wlr().
- Compute sample size and bounds using gs_design_npe().
- Return a list of design enrollment, failure rates, and bounds.

Examples

```
library(mvtnorm)
library(gsDesign)
library(gsDesign2)
```

```
# set enrollment rates
enroll_rate <- define_enroll_rate(duration = 12, rate = 1)

# set failure rates
fail_rate <- define_fail_rate(
  duration = c(4, 100),
  fail_rate = log(2) / 15, # median survival 15 month
  hr = c(1, .6),
  dropout_rate = 0.001
)

# Example 1 ----
# Information fraction driven design
gs_design_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  ratio = 1,
  alpha = 0.025, beta = 0.2,
  weight = list(method = "mb", param = list(tau = Inf, w_max = 2)),
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.2),
  analysis_time = 36,
  info_frac = c(0.6, 1)
)

# Example 2 ----
# Calendar time driven design
gs_design_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  ratio = 1,
  alpha = 0.025, beta = 0.2,
  weight = list(method = "mb", param = list(tau = Inf, w_max = 2)),
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.2),
  analysis_time = c(24, 36),
  info_frac = NULL
)

# Example 3 ----
# Both calendar time and information fraction driven design
gs_design_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  ratio = 1,
  alpha = 0.025, beta = 0.2,
  weight = list(method = "mb", param = list(tau = Inf, w_max = 2)),
  upper = gs_spending_bound,
```

```
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.2),
  analysis_time = c(24, 36),
  info_frac = c(0.6, 1)
)
```

gs_info_ahr *Information and effect size based on AHR approximation*

Description

Based on piecewise enrollment rate, failure rate, and dropout rates computes approximate information and effect size using an average hazard ratio model.

Usage

```
gs_info_ahr(
  enroll_rate = define_enroll_rate(duration = c(2, 2, 10), rate = c(3, 6, 9)),
 fail_rate = define_fail_rate(duration = c(3, 100), fail_rate = log(2)/c(9, 18), hr =
    c(0.9, 0.6), dropout_rate = 0.001),
  ratio = 1,
  event = NULL,
  analysis_time = NULL,
  interval = c(0.01, 1000)
)
```

Arguments

enroll_rate	Enrollment rates from define_enroll_rate().
fail_rate	Failure and dropout rates from define_fail_rate().
ratio	Experimental:Control randomization ratio.
event	Targeted minimum events at each analysis.
analysis_time	Targeted minimum study duration at each analysis.
interval	An interval that is presumed to include the time at which expected event count is equal to targeted event.

Details

The ahr() function computes statistical information at targeted event times. The expected_time() function is used to get events and average HR at targeted analysis_time.

Value

A data frame with columns analysis, time, ahr, event, theta, info, info0. The columns info and info0 contain statistical information under H1, H0, respectively. For analysis k, time[k] is the maximum of analysis_time[k] and the expected time required to accrue the targeted event[k]. ahr is the expected average hazard ratio at each analysis.

Specification

- Validate if input event is a numeric value vector or a vector with increasing values.
- Validate if input analysis_time is a numeric value vector or a vector with increasing values.
- Validate if inputs event and analysis_time have the same length if they are both specified.
- Compute average hazard ratio:
  - If analysis_time is specified, calculate average hazard ratio using `ahr()`.
  - If event is specified, calculate average hazard ratio using `expected_time()`.
- Return a data frame of Analysis, Time, AHR, Events, theta, info, info0.

Examples

```
library(gsDesign)
library(gsDesign2)

# Example 1 ----

# Only put in targeted events
gs_info_ahr(event = c(30, 40, 50))


# Example 2 ----
# Only put in targeted analysis times
gs_info_ahr(analysis_time = c(18, 27, 36))

# Example 3 ----

# Some analysis times after time at which targeted event accrue
# Check that both Time >= input analysis_time and event >= input event
gs_info_ahr(event = c(30, 40, 50), analysis_time = c(16, 19, 26))
gs_info_ahr(event = c(30, 40, 50), analysis_time = c(14, 20, 24))
```

`gs_info_combo` *Information and effect size for MaxCombo test*

Description

Information and effect size for MaxCombo test

Usage

```
gs_info_combo(
  enroll_rate = define_enroll_rate(duration = c(2, 2, 10), rate = c(3, 6, 9)),
  fail_rate = define_fail_rate(duration = c(3, 100), fail_rate = log(2)/c(9, 18), hr =
    c(0.9, 0.6), dropout_rate = 0.001),
  ratio = 1,
  event = NULL,
```

```
    analysis_time = NULL,
    rho,
    gamma,
    tau = rep(-1, length(rho)),
    approx = "asymptotic"
  )
```

Arguments

enroll_rate	An enroll_rate data frame with or without stratum created by define_enroll_rate().
fail_rate	A fail_rate data frame with or without stratum created by define_fail_rate().
ratio	Experimental:Control randomization ratio (not yet implemented).
event	Targeted events at each analysis.
analysis_time	Minimum time of analysis.
rho	Weighting parameters.
gamma	Weighting parameters.
tau	Weighting parameters.
approx	Approximation method.

Value

A tibble with columns as test index, analysis index, analysis time, sample size, number of events, ahr, delta, sigma2, theta, and statistical information.

Examples

```
gs_info_combo(rho = c(0, 0.5), gamma = c(0.5, 0), analysis_time = c(12, 24))
```

---

gs_info_rd *Information and effect size under risk difference*

Description

Information and effect size under risk difference

Usage

```
gs_info_rd(
  p_c = tibble::tibble(stratum = "All", rate = 0.2),
  p_e = tibble::tibble(stratum = "All", rate = 0.15),
  n = tibble::tibble(stratum = "All", n = c(100, 200, 300), analysis = 1:3),
  rd0 = 0,
  ratio = 1,
  weight = c("unstratified", "ss", "invar", "mr")
)
```

Arguments

p_c	Rate at the control group.
p_e	Rate at the experimental group.
n	Sample size.
rd0	The risk difference under H0.
ratio	Experimental:Control randomization ratio.
weight	Weighting method, can be "unstratified", "ss", or "invar".

Value

A tibble with columns as analysis index, sample size, risk difference, risk difference under null hypothesis, theta1 (standardized treatment effect under alternative hypothesis), theta0 (standardized treatment effect under null hypothesis), and statistical information.

Examples

```
# Example 1 ----
# unstratified case with H0: rd0 = 0
gs_info_rd(
  p_c = tibble::tibble(stratum = "All", rate = .15),
  p_e = tibble::tibble(stratum = "All", rate = .1),
  n = tibble::tibble(stratum = "All", n = c(100, 200, 300), analysis = 1:3),
  rd0 = 0,
  ratio = 1
)

# Example 2 ----
# unstratified case with H0: rd0 != 0
gs_info_rd(
  p_c = tibble::tibble(stratum = "All", rate = .2),
  p_e = tibble::tibble(stratum = "All", rate = .15),
  n = tibble::tibble(stratum = "All", n = c(100, 200, 300), analysis = 1:3),
  rd0 = 0.005,
  ratio = 1
)

# Example 3 ----
# stratified case under sample size weighting and H0: rd0 = 0
gs_info_rd(
  p_c = tibble::tibble(stratum = c("S1", "S2", "S3"), rate = c(.15, .2, .25)),
  p_e = tibble::tibble(stratum = c("S1", "S2", "S3"), rate = c(.1, .16, .19)),
  n = tibble::tibble(
    stratum = rep(c("S1", "S2", "S3"), each = 3),
    analysis = rep(1:3, 3),
    n = c(50, 100, 200, 40, 80, 160, 60, 120, 240)
  ),
  rd0 = 0,
  ratio = 1,
  weight = "ss"
)
```

```
# Example 4 ----
# stratified case under inverse variance weighting and H0: rd0 = 0
gs_info_rd(
  p_c = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.15, .2, .25)
  ),
  p_e = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.1, .16, .19)
  ),
  n = tibble::tibble(
    stratum = rep(c("S1", "S2", "S3"), each = 3),
    analysis = rep(1:3, 3),
    n = c(50, 100, 200, 40, 80, 160, 60, 120, 240)
  ),
  rd0 = 0,
  ratio = 1,
  weight = "invar"
)

# Example 5 ----
# stratified case under minimal risk weighting and H0: rd0 = 0
gs_info_rd(
  p_c = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.15, .2, .25)
  ),
  p_e = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.1, .16, .19)
  ),
  n = tibble::tibble(
    stratum = rep(c("S1", "S2", "S3"), each = 3),
    analysis = rep(1:3, 3),
    n = c(50, 100, 200, 40, 80, 160, 60, 120, 240)
  ),
  rd0 = 0,
  ratio = 1,
  weight = "mr"
)

# Example 6 ----
# stratified case under sample size weighting and H0: rd0 != 0
gs_info_rd(
  p_c = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.15, .2, .25)
  ),
  p_e = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.1, .16, .19)
```

```
  ),
  n = tibble::tibble(
    stratum = rep(c("S1", "S2", "S3"), each = 3),
    analysis = rep(1:3, 3),
    n = c(50, 100, 200, 40, 80, 160, 60, 120, 240)
  ),
  rd0 = 0.02,
  ratio = 1,
  # users can switch to either "invar" or "mr" weighting as well
  weight = "ss"
)

# Example 7 ----
# stratified case under inverse variance weighting and H0: rd0 != 0 and
# rd0 difference for different stratum
gs_info_rd(
  p_c = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.15, .2, .25)
  ),
  p_e = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.1, .16, .19)
  ),
  n = tibble::tibble(
    stratum = rep(c("S1", "S2", "S3"), each = 3),
    analysis = rep(1:3, 3),
    n = c(50, 100, 200, 40, 80, 160, 60, 120, 240)
  ),
  rd0 = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rd0 = c(0.01, 0.02, 0.03)
  ),
  ratio = 1,
  weight = "invar"
)
```

`gs_info_wlr` *Information and effect size for weighted log-rank test*

Description

Based on piecewise enrollment rate, failure rate, and dropout rates computes approximate information and effect size using an average hazard ratio model.

Usage

```
gs_info_wlr(
  enroll_rate = define_enroll_rate(duration = c(2, 2, 10), rate = c(3, 6, 9)),
  fail_rate = define_fail_rate(duration = c(3, 100), fail_rate = log(2)/c(9, 18), hr =
```

```
    c(0.9, 0.6), dropout_rate = 0.001),
  ratio = 1,
  event = NULL,
  analysis_time = NULL,
  weight = "logrank",
  approx = "asymptotic",
  interval = c(0.01, 1000)
)
```

Arguments

`enroll_rate`	An `enroll_rate` data frame with or without stratum created by `define_enroll_rate()`.
`fail_rate`	Failure and dropout rates.
`ratio`	Experimental:Control randomization ratio.
`event`	Targeted minimum events at each analysis.
`analysis_time`	Targeted minimum study duration at each analysis.
`weight`	Weight of weighted log rank test: • `"logrank"` = regular logrank test. • `list(method = "fh", param = list(rho = ..., gamma = ...))` = Fleming-Harrington weighting functions. • `list(method = "mb", param = list(tau = ..., w_max = ...))` = Magirr and Burman weighting functions.
`approx`	Approximate estimation method for Z statistics. • `"event_driven"` = only work under proportional hazard model with log rank test. • `"asymptotic"`.
`interval`	An interval that is presumed to include the time at which expected event count is equal to targeted event.

Details

The `ahr()` function computes statistical information at targeted event times. The `expected_time()` function is used to get events and average HR at targeted `analysis_time`.

Value

A tibble with columns Analysis, Time, N, Events, AHR, delta, sigma2, theta, info, info0. `info` and `info0` contain statistical information under H1, H0, respectively. For analysis `k`, `Time[k]` is the maximum of `analysis_time[k]` and the expected time required to accrue the targeted `event[k]`. `AHR` is the expected average hazard ratio at each analysis.

Examples

```
library(gsDesign2)

# Set enrollment rates
```

```
enroll_rate <- define_enroll_rate(duration = 12, rate = 500 / 12)

# Set failure rates
fail_rate <- define_fail_rate(
  duration = c(4, 100),
  fail_rate = log(2) / 15, # median survival 15 month
  hr = c(1, .6),
  dropout_rate = 0.001
)

# Set the targeted number of events and analysis time
event <- c(30, 40, 50)
analysis_time <- c(10, 24, 30)

gs_info_wlr(
  enroll_rate = enroll_rate, fail_rate = fail_rate,
  event = event, analysis_time = analysis_time
)
```

gs_power_ahr — *Group sequential design power using average hazard ratio under non-proportional hazards*

Description

Calculate power given the sample size in group sequential design power using average hazard ratio under non-proportional hazards.

Usage

```
gs_power_ahr(
  enroll_rate = define_enroll_rate(duration = c(2, 2, 10), rate = c(3, 6, 9)),
  fail_rate = define_fail_rate(duration = c(3, 100), fail_rate = log(2)/c(9, 18), hr =
    c(0.9, 0.6), dropout_rate = rep(0.001, 2)),
  event = c(30, 40, 50),
  analysis_time = NULL,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = NULL),
  test_lower = TRUE,
  test_upper = TRUE,
  ratio = 1,
  binding = FALSE,
  h1_spending = TRUE,
  info_scale = c("h0_h1_info", "h0_info", "h1_info"),
  r = 18,
  tol = 1e-06,
```

```
  interval = c(0.01, 1000),
  integer = FALSE
)
```

Arguments

- `enroll_rate` An `enroll_rate` data frame with or without stratum created by `define_enroll_rate()`.
- `fail_rate` A `fail_rate` data frame with or without stratum created by `define_fail_rate()`.
- `event` A numerical vector specifying the targeted events at each analysis. See details.
- `analysis_time` Targeted calendar timing of analyses. See details.
- `upper` Function to compute upper bound.
  - `gs_spending_bound()`: alpha-spending efficacy bounds.
  - `gs_b()`: fixed efficacy bounds.
- `upar` Parameters passed to `upper`.
  - If `upper = gs_b`, then `upar` is a numerical vector specifying the fixed efficacy bounds per analysis.
  - If `upper = gs_spending_bound`, then `upar` is a list including
    - `sf` for the spending function family.
    - `total_spend` for total alpha spend.
    - `param` for the parameter of the spending function.
    - `timing` specifies spending time if different from information-based spending; see details.
- `lower` Function to compute lower bound, which can be set up similarly as `upper`. See this vignette.
- `lpar` Parameters passed to `lower`, which can be set up similarly as `upar`.
- `test_lower` Indicator of which analyses should include a lower bound; single value of `TRUE` (default) indicates all analyses; single value of `FALSE` indicated no lower bound; otherwise, a logical vector of the same length as `info` should indicate which analyses will have a lower bound.
- `test_upper` Indicator of which analyses should include an upper (efficacy) bound; single value of `TRUE` (default) indicates all analyses; otherwise, a logical vector of the same length as `info` should indicate which analyses will have an efficacy bound.
- `ratio` Experimental:Control randomization ratio.
- `binding` Indicator of whether futility bound is binding; default of `FALSE` is recommended.
- `h1_spending` Indicator that lower bound to be set by spending under alternate hypothesis (input `fail_rate`) if spending is used for lower bound. If this is `FALSE`, then the lower bound spending is under the null hypothesis. This is for two-sided symmetric or asymmetric testing under the null hypothesis; See this vignette.
- `info_scale` Information scale for calculation. Options are:
  - `"h0_h1_info"` (default): variance under both null and alternative hypotheses is used.
  - `"h0_info"`: variance under null hypothesis is used. This is often used for testing methods that use local alternatives, such as the Schoenfeld method.

- "h1_info": variance under alternative hypothesis is used.

r
: Integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally, r will not be changed by the user.

tol
: Tolerance parameter for boundary convergence (on Z-scale); normally not changed by the user.

interval
: An interval presumed to include the times at which expected event count is equal to targeted event. Normally, this can be ignored by the user as it is set to c(.01, 1000).

integer
: Indicator of whether integer sample size and events are intended. This argument is used when using to_integer().

Details

Note that time units are arbitrary, but should be the same for all rate parameters in enroll_rate, fail_rate, and analysis_time.

Computed bounds satisfy input upper bound specification in upper, upar, and lower bound specification in lower, lpar. ahr() computes statistical information at targeted event times. The expected_time() function is used to get events and average HR at targeted analysis_time.

The parameters event and analysis_time are used to determine the timing for interim and final analyses.

- If analysis timing is to be determined by targeted events, then event is a numerical vector specifying the targeted events for each analysis; note that this can be NULL.
- If interim analysis is determined by targeted calendar timing relative to start of enrollment, then analysis_time will be a vector specifying the calendar time from start of study for each analysis; note that this can be NULL.
- A corresponding element of event or analysis_time should be provided for each analysis.
- If both event[i] and analysis[i] are provided for analysis i, then the time corresponding to the later of these is used for analysis i.

Value

A list with input parameters, enrollment rate, analysis, and bound.

- $input a list including alpha, beta, ratio, etc.
- $enroll_rate a table showing the enrollment, which is the same as input.
- $fail_rate a table showing the failure and dropout rates, which is the same as input.
- $bound a table summarizing the efficacy and futility bound at each analysis.
- analysis a table summarizing the analysis time, sample size, events, average HR, treatment effect and statistical information at each analysis.

Specification

- Calculate information and effect size based on AHR approximation using gs_info_ahr().
- Return a tibble of with columns Analysis, Bound, Z, Probability, theta, Time, AHR, Events and contains a row for each analysis and each bound.

Examples

```
library(gsDesign2)

# Example 1 ----
# The default output of `gs_power_ahr()` is driven by events,
# i.e., `event = c(30, 40, 50)`, `analysis_time = NULL`

gs_power_ahr(lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.1))

# Example 2 ----
# 2-sided symmetric O'Brien-Fleming spending bound, driven by analysis time,
# i.e., `event = NULL`, `analysis_time = c(12, 24, 36)`

gs_power_ahr(
  analysis_time = c(12, 24, 36),
  event = NULL,
  binding = TRUE,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.025)
)

# Example 3 ----
# 2-sided symmetric O'Brien-Fleming spending bound, driven by event,
# i.e., `event = c(20, 50, 70)`, `analysis_time = NULL`
# Note that this assumes targeted final events for the design is 70 events.
# If actual targeted final events were 65, then `timing = c(20, 50, 70) / 65`
# would be added to `upar` and `lpar` lists.
# NOTE: at present the computed information fraction in output `analysis` is based
# on 70 events rather than 65 events when the `timing` argument is used in this way.
# A vignette on this topic will be forthcoming.

gs_power_ahr(
  analysis_time = NULL,
  event = c(20, 50, 70),
  binding = TRUE,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.025)
)

# Example 4 ----
# 2-sided symmetric O'Brien-Fleming spending bound,
# driven by both `event` and `analysis_time`, i.e.,
# both `event` and `analysis_time` are not `NULL`,
# then the analysis will driven by the maximal one, i.e.,
# Time = max(analysis_time, calculated Time for targeted event)
# Events = max(events, calculated events for targeted analysis_time)

gs_power_ahr(
```

```
  analysis_time = c(12, 24, 36),
  event = c(30, 40, 50), h1_spending = FALSE,
  binding = TRUE,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.025)
)
```

---

gs_power_combo — *Group sequential design power using MaxCombo test under non-proportional hazards*

Description

Group sequential design power using MaxCombo test under non-proportional hazards

Usage

```
gs_power_combo(
  enroll_rate = define_enroll_rate(duration = 12, rate = 500/12),
  fail_rate = define_fail_rate(duration = c(4, 100), fail_rate = log(2)/15, hr = c(1,
    0.6), dropout_rate = 0.001),
  fh_test = rbind(data.frame(rho = 0, gamma = 0, tau = -1, test = 1, analysis = 1:3,
   analysis_time = c(12, 24, 36)), data.frame(rho = c(0, 0.5), gamma = 0.5, tau = -1,
    test = 2:3, analysis = 3, analysis_time = 36)),
  ratio = 1,
  binding = FALSE,
  upper = gs_b,
  upar = c(3, 2, 1),
  lower = gs_b,
  lpar = c(-1, 0, 1),
  algorithm = mvtnorm::GenzBretz(maxpts = 1e+05, abseps = 1e-05),
  ...
)
```

Arguments

enroll_rate	An enroll_rate data frame with or without stratum created by define_enroll_rate().
fail_rate	A fail_rate data frame with or without stratum created by define_fail_rate().
fh_test	A data frame to summarize the test in each analysis. See examples for its data structure.
ratio	Experimental:Control randomization ratio.
binding	Indicator of whether futility bound is binding; default of FALSE is recommended.
upper	Function to compute upper bound.

- `gs_spending_bound()`: alpha-spending efficacy bounds.
- `gs_b()`: fixed efficacy bounds.

`upar` Parameters passed to `upper`.

- If `upper = gs_b`, then `upar` is a numerical vector specifying the fixed efficacy bounds per analysis.
- If `upper = gs_spending_bound`, then `upar` is a list including
  - `sf` for the spending function family.
  - `total_spend` for total alpha spend.
  - `param` for the parameter of the spending function.
  - `timing` specifies spending time if different from information-based spending; see details.

`lower` Function to compute lower bound, which can be set up similarly as `upper`. See this vignette.

`lpar` Parameters passed to `lower`, which can be set up similarly as `upar`.

`algorithm` an object of class `GenzBretz`, `Miwa` or `TVPACK` specifying both the algorithm to be used as well as the associated hyper parameters.

`...` Additional parameters passed to mvtnorm::pmvnorm.

Value

A list with input parameters, enrollment rate, analysis, and bound.

Specification

- Validate if lower and upper bounds have been specified.
- Extract info, info_fh, theta_fh and corr_fh from utility.
- Extract sample size via the maximum sample size of info.
- Calculate information fraction either for fixed or group sequential design.
- Compute spending function using `gs_bound()`.
- Compute probability of crossing bounds under the null and alternative hypotheses using `gs_prob_combo()`.
- Export required information for boundary and crossing probability

Examples

```
library(mvtnorm)
library(gsDesign)
library(gsDesign2)

enroll_rate <- define_enroll_rate(
  duration = 12,
  rate = 500 / 12
)

fail_rate <- define_fail_rate(
  duration = c(4, 100),
```

```
    fail_rate = log(2) / 15, # median survival 15 month
    hr = c(1, .6),
    dropout_rate = 0.001
  )

  fh_test <- rbind(
   data.frame(rho = 0, gamma = 0, tau = -1, test = 1, analysis = 1:3, analysis_time = c(12, 24, 36)),
   data.frame(rho = c(0, 0.5), gamma = 0.5, tau = -1, test = 2:3, analysis = 3, analysis_time = 36)
  )

  # Example 1 ----
  # Minimal Information Fraction derived bound

  gs_power_combo(
    enroll_rate = enroll_rate,
    fail_rate = fail_rate,
    fh_test = fh_test,
    upper = gs_spending_combo,
    upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
    lower = gs_spending_combo,
    lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.2)
  )
```

---

gs_power_rd *Group sequential design power of binary outcome measuring in risk difference*

Description

Group sequential design power of binary outcome measuring in risk difference

Usage

```
gs_power_rd(
  p_c = tibble::tibble(stratum = "All", rate = 0.2),
  p_e = tibble::tibble(stratum = "All", rate = 0.15),
  n = tibble::tibble(stratum = "All", n = c(40, 50, 60), analysis = 1:3),
  rd0 = 0,
  ratio = 1,
  weight = c("unstratified", "ss", "invar", "mr"),
  upper = gs_b,
  lower = gs_b,
  upar = gsDesign(k = 3, test.type = 1, sfu = sfLDOF, sfupar = NULL)$upper$bound,
  lpar = c(qnorm(0.1), rep(-Inf, 2)),
  info_scale = c("h0_h1_info", "h0_info", "h1_info"),
  binding = FALSE,
  test_upper = TRUE,
  test_lower = TRUE,
```

```
    r = 18,
    tol = 1e-06
  )
```

Arguments

p_c	Rate at the control group.
p_e	Rate at the experimental group.
n	Sample size.
rd0	Treatment effect under super-superiority designs, the default is 0.
ratio	Experimental:control randomization ratio.
weight	Weighting method, can be "unstratified", "ss", or "invar".
upper	Function to compute upper bound.
lower	Function to compare lower bound.
upar	Parameters passed to upper.
lpar	Parameters passed to lower.
info_scale	Information scale for calculation. Options are: • "h0_h1_info" (default): variance under both null and alternative hypotheses is used. • "h0_info": variance under null hypothesis is used. • "h1_info": variance under alternative hypothesis is used.
binding	Indicator of whether futility bound is binding; default of FALSE is recommended.
test_upper	Indicator of which analyses should include an upper (efficacy) bound; single value of TRUE (default) indicates all analyses; otherwise, a logical vector of the same length as info should indicate which analyses will have an efficacy bound.
test_lower	Indicator of which analyses should include a lower bound; single value of TRUE (default) indicates all analyses; single value FALSE indicated no lower bound; otherwise, a logical vector of the same length as info should indicate which analyses will have a lower bound.
r	Integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally, r will not be changed by the user.
tol	Tolerance parameter for boundary convergence (on Z-scale).

Value

A list with input parameter, analysis, and bound.

Examples

```
# Example 1 ----
library(gsDesign)

# unstratified case with H0: rd0 = 0
```

```
gs_power_rd(
  p_c = tibble::tibble(
    stratum = "All",
    rate = .2
  ),
  p_e = tibble::tibble(
    stratum = "All",
    rate = .15
  ),
  n = tibble::tibble(
    stratum = "All",
    n = c(20, 40, 60),
    analysis = 1:3
  ),
  rd0 = 0,
  ratio = 1,
  upper = gs_b,
  lower = gs_b,
  upar = gsDesign(k = 3, test.type = 1, sfu = sfLDOF, sfupar = NULL)$upper$bound,
  lpar = c(qnorm(.1), rep(-Inf, 2))
)

# Example 2 ----
# unstratified case with H0: rd0 != 0
gs_power_rd(
  p_c = tibble::tibble(
    stratum = "All",
    rate = .2
  ),
  p_e = tibble::tibble(
    stratum = "All",
    rate = .15
  ),
  n = tibble::tibble(
    stratum = "All",
    n = c(20, 40, 60),
    analysis = 1:3
  ),
  rd0 = 0.005,
  ratio = 1,
  upper = gs_b,
  lower = gs_b,
  upar = gsDesign(k = 3, test.type = 1, sfu = sfLDOF, sfupar = NULL)$upper$bound,
  lpar = c(qnorm(.1), rep(-Inf, 2))
)

# use spending function
gs_power_rd(
  p_c = tibble::tibble(
    stratum = "All",
    rate = .2
  ),
  p_e = tibble::tibble(
```

```
    stratum = "All",
    rate = .15
  ),
  n = tibble::tibble(
    stratum = "All",
    n = c(20, 40, 60),
    analysis = 1:3
  ),
  rd0 = 0.005,
  ratio = 1,
  upper = gs_spending_bound,
  lower = gs_b,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lpar = c(qnorm(.1), rep(-Inf, 2))
)

# Example 3 ----
# stratified case under sample size weighting and H0: rd0 = 0
gs_power_rd(
  p_c = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.15, .2, .25)
  ),
  p_e = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.1, .16, .19)
  ),
  n = tibble::tibble(
    stratum = rep(c("S1", "S2", "S3"), each = 3),
    analysis = rep(1:3, 3),
    n = c(10, 20, 24, 18, 26, 30, 10, 20, 24)
  ),
  rd0 = 0,
  ratio = 1,
  weight = "ss",
  upper = gs_b,
  lower = gs_b,
  upar = gsDesign(k = 3, test.type = 1, sfu = sfLDOF, sfupar = NULL)$upper$bound,
  lpar = c(qnorm(.1), rep(-Inf, 2))
)

# Example 4 ----
# stratified case under inverse variance weighting and H0: rd0 = 0
gs_power_rd(
  p_c = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.15, .2, .25)
  ),
  p_e = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.1, .16, .19)
  ),
  n = tibble::tibble(
```

```
    stratum = rep(c("S1", "S2", "S3"), each = 3),
    analysis = rep(1:3, 3),
    n = c(10, 20, 24, 18, 26, 30, 10, 20, 24)
  ),
  rd0 = 0,
  ratio = 1,
  weight = "invar",
  upper = gs_b,
  lower = gs_b,
  upar = gsDesign(k = 3, test.type = 1, sfu = sfLDOF, sfupar = NULL)$upper$bound,
  lpar = c(qnorm(.1), rep(-Inf, 2))
)

# Example 5 ----
# stratified case under sample size weighting and H0: rd0 != 0
gs_power_rd(
  p_c = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.15, .2, .25)
  ),
  p_e = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.1, .16, .19)
  ),
  n = tibble::tibble(
    stratum = rep(c("S1", "S2", "S3"), each = 3),
    analysis = rep(1:3, 3),
    n = c(10, 20, 24, 18, 26, 30, 10, 20, 24)
  ),
  rd0 = 0.02,
  ratio = 1,
  weight = "ss",
  upper = gs_b,
  lower = gs_b,
  upar = gsDesign(k = 3, test.type = 1, sfu = sfLDOF, sfupar = NULL)$upper$bound,
  lpar = c(qnorm(.1), rep(-Inf, 2))
)

# Example 6 ----
# stratified case under inverse variance weighting and H0: rd0 != 0
gs_power_rd(
  p_c = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.15, .2, .25)
  ),
  p_e = tibble::tibble(
    stratum = c("S1", "S2", "S3"),
    rate = c(.1, .16, .19)
  ),
  n = tibble::tibble(
    stratum = rep(c("S1", "S2", "S3"), each = 3),
    analysis = rep(1:3, 3),
    n = c(10, 20, 24, 18, 26, 30, 10, 20, 24)
```

```
  ),
  rd0 = 0.03,
  ratio = 1,
  weight = "invar",
  upper = gs_b,
  lower = gs_b,
  upar = gsDesign(k = 3, test.type = 1, sfu = sfLDOF, sfupar = NULL)$upper$bound,
  lpar = c(qnorm(.1), rep(-Inf, 2))
)
```

---

gs_power_wlr — *Group sequential design power using weighted log rank test under non-proportional hazards*

Description

Group sequential design power using weighted log rank test under non-proportional hazards

Usage

```
gs_power_wlr(
  enroll_rate = define_enroll_rate(duration = c(2, 2, 10), rate = c(3, 6, 9)),
 fail_rate = tibble(stratum = "All", duration = c(3, 100), fail_rate = log(2)/c(9, 18),
    hr = c(0.9, 0.6), dropout_rate = rep(0.001, 2)),
  event = c(30, 40, 50),
  analysis_time = NULL,
  binding = FALSE,
  h1_spending = TRUE,
  upper = gs_spending_bound,
  lower = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
  lpar = list(sf = gsDesign::sfLDOF, total_spend = NULL),
  test_upper = TRUE,
  test_lower = TRUE,
  ratio = 1,
  weight = "logrank",
  info_scale = c("h0_h1_info", "h0_info", "h1_info"),
  approx = "asymptotic",
  r = 18,
  tol = 1e-06,
  interval = c(0.01, 1000),
  integer = FALSE
)
```

Arguments

enroll_rate	An enroll_rate data frame with or without stratum created by define_enroll_rate().
fail_rate	A fail_rate data frame with or without stratum created by define_fail_rate().

event — A numerical vector specifying the targeted events at each analysis. See details.

analysis_time — Targeted calendar timing of analyses. See details.

binding — Indicator of whether futility bound is binding; default of FALSE is recommended.

h1_spending — Indicator that lower bound to be set by spending under alternate hypothesis (input fail_rate) if spending is used for lower bound. If this is FALSE, then the lower bound spending is under the null hypothesis. This is for two-sided symmetric or asymmetric testing under the null hypothesis; See this vignette.

upper — Function to compute upper bound.

- gs_spending_bound(): alpha-spending efficacy bounds.
- gs_b(): fixed efficacy bounds.

lower — Function to compute lower bound, which can be set up similarly as upper. See this vignette.

upar — Parameters passed to upper.

- If upper = gs_b, then upar is a numerical vector specifying the fixed efficacy bounds per analysis.
- If upper = gs_spending_bound, then upar is a list including
  - sf for the spending function family.
  - total_spend for total alpha spend.
  - param for the parameter of the spending function.
  - timing specifies spending time if different from information-based spending; see details.

lpar — Parameters passed to lower, which can be set up similarly as upar.

test_upper — Indicator of which analyses should include an upper (efficacy) bound; single value of TRUE (default) indicates all analyses; otherwise, a logical vector of the same length as info should indicate which analyses will have an efficacy bound.

test_lower — Indicator of which analyses should include a lower bound; single value of TRUE (default) indicates all analyses; single value of FALSE indicated no lower bound; otherwise, a logical vector of the same length as info should indicate which analyses will have a lower bound.

ratio — Experimental:Control randomization ratio.

weight — Weight of weighted log rank test:

- "logrank" = regular logrank test.
- list(method = "fh", param = list(rho = ..., gamma = ...)) = Fleming-Harrington weighting functions.
- list(method = "mb", param = list(tau = ..., w_max = ...)) = Magirr and Burman weighting functions.

info_scale — Information scale for calculation. Options are:

- "h0_h1_info" (default): variance under both null and alternative hypotheses is used.
- "h0_info": variance under null hypothesis is used. This is often used for testing methods that use local alternatives, such as the Schoenfeld method.
- "h1_info": variance under alternative hypothesis is used.

approx
: Approximate estimation method for Z statistics.
  - "event_driven" = only work under proportional hazard model with log rank test.
  - "asymptotic".

r
: Integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally, r will not be changed by the user.

tol
: Tolerance parameter for boundary convergence (on Z-scale); normally not changed by the user.

interval
: An interval presumed to include the times at which expected event count is equal to targeted event. Normally, this can be ignored by the user as it is set to c(.01, 1000).

integer
: Indicator of whether integer sample size and events are intended. This argument is used when using to_integer().

Value

A list with input parameters, enrollment rate, analysis, and bound.

Specification

- Compute information and effect size for Weighted Log-rank test using gs_info_wlr().
- Compute group sequential bound computation with non-constant effect using gs_power_npe().
- Combine information and effect size and power and return a tibble with columns Analysis, Bound, Time, Events, Z, Probability, AHR, theta, info, and info0.

Examples

```
library(gsDesign)
library(gsDesign2)

# set enrollment rates
enroll_rate <- define_enroll_rate(duration = 12, rate = 500 / 12)

# set failure rates
fail_rate <- define_fail_rate(
  duration = c(4, 100),
  fail_rate = log(2) / 15, # median survival 15 month
  hr = c(1, .6),
  dropout_rate = 0.001
)

# set the targeted number of events and analysis time
target_events <- c(30, 40, 50)
target_analysisTime <- c(10, 24, 30)

# Example 1 ----

# fixed bounds and calculate the power for targeted number of events
```

```
gs_power_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  event = target_events,
  analysis_time = NULL,
  upper = gs_b,
  upar = gsDesign(
    k = length(target_events),
    test.type = 1,
    n.I = target_events,
    maxn.IPlan = max(target_events),
    sfu = sfLDOF,
    sfupar = NULL
  )$upper$bound,
  lower = gs_b,
  lpar = c(qnorm(.1), rep(-Inf, 2))
)

# Example 2 ----
# fixed bounds and calculate the power for targeted analysis time

gs_power_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  event = NULL,
  analysis_time = target_analysisTime,
  upper = gs_b,
  upar = gsDesign(
    k = length(target_events),
    test.type = 1,
    n.I = target_events,
    maxn.IPlan = max(target_events),
    sfu = sfLDOF,
    sfupar = NULL
  )$upper$bound,
  lower = gs_b,
  lpar = c(qnorm(.1), rep(-Inf, 2))
)

# Example 3 ----
# fixed bounds and calculate the power for targeted analysis time & number of events

gs_power_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  event = target_events,
  analysis_time = target_analysisTime,
  upper = gs_b,
  upar = gsDesign(
    k = length(target_events),
    test.type = 1,
    n.I = target_events,
    maxn.IPlan = max(target_events),
```

```
    sfu = sfLDOF,
    sfupar = NULL
  )$upper$bound,
  lower = gs_b,
  lpar = c(qnorm(.1), rep(-Inf, 2))
)

# Example 4 ----
# spending bounds and calculate the power for targeted number of events

gs_power_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  event = target_events,
  analysis_time = NULL,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.2)
)

# Example 5 ----
# spending bounds and calculate the power for targeted analysis time

gs_power_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  event = NULL,
  analysis_time = target_analysisTime,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.2)
)

# Example 6 ----
# spending bounds and calculate the power for targeted analysis time & number of events

gs_power_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  event = target_events,
  analysis_time = target_analysisTime,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.2)
)
```

gs_spending_bound *Derive spending bound for group sequential boundary*

Description

Computes one bound at a time based on spending under given distributional assumptions. While user specifies `gs_spending_bound()` for use with other functions, it is not intended for use on its own. Most important user specifications are made through a list provided to functions using `gs_spending_bound()`. Function uses numerical integration and Newton-Raphson iteration to derive an individual bound for a group sequential design that satisfies a targeted boundary crossing probability. Algorithm is a simple extension of that in Chapter 19 of Jennison and Turnbull (2000).

Usage

```
gs_spending_bound(
  k = 1,
  par = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL,
    max_info = NULL),
  hgm1 = NULL,
  theta = 0.1,
  info = 1:3,
  efficacy = TRUE,
  test_bound = TRUE,
  r = 18,
  tol = 1e-06
)
```

Arguments

`k`	Analysis for which bound is to be computed.
`par`	A list with the following items: • `sf` (class spending function). • `total_spend` (total spend). • `param` (any parameters needed by the spending function `sf()`). • `timing` (a vector containing values at which spending function is to be evaluated or `NULL` if information-based spending is used). • `max_info` (when `timing` is `NULL`, this can be input as positive number to be used with `info` for information fraction at each analysis).
`hgm1`	Subdensity grid from `h1()` (k=2) or `hupdate()` (k>2) for analysis k-1; if k=1, this is not used and may be `NULL`.
`theta`	Natural parameter used for lower bound only spending; represents average drift at each time of analysis at least up to analysis k; upper bound spending is always set under null hypothesis (theta = 0).
`info`	Statistical information at all analyses, at least up to analysis k.
`efficacy`	`TRUE` (default) for efficacy bound, `FALSE` otherwise.
`test_bound`	A logical vector of the same length as `info` should indicate which analyses will have a bound.
`r`	Integer value controlling grid for numerical integration as in Jennison and Turnbull (2000); default is 18, range is 1 to 80. Larger values provide larger number of grid points and greater accuracy. Normally `r` will not be changed by the user.
`tol`	Tolerance parameter for convergence (on Z-scale).

Value

Returns a numeric bound (possibly infinite) or, upon failure, generates an error message.

Specification

- Set the spending time at analysis.
- Compute the cumulative spending at analysis.
- Compute the incremental spend at each analysis.
- Set test_bound a vector of length k > 1 if input as a single value.
- Compute spending for current bound.
- Iterate to convergence as in gsbound.c from gsDesign.
- Compute subdensity for final analysis in rejection region.
- Validate the output and return an error message in case of failure.
- Return a numeric bound (possibly infinite).

Author(s)

Keaven Anderson <keaven_anderson@merck.com>

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

Examples

```
gs_power_ahr(
  analysis_time = c(12, 24, 36),
  event = c(30, 40, 50),
  binding = TRUE,
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL)
)
```

---

gs_spending_combo   *Derive spending bound for MaxCombo group sequential boundary*

Description

Derive spending bound for MaxCombo group sequential boundary

Usage

```
gs_spending_combo(par = NULL, info = NULL)
```

Arguments

par
: A list with the following items:
  - `sf` (class spending function).
  - `total_spend` (total spend).
  - `param` (any parameters needed by the spending function `sf()`).
  - `timing` (a vector containing values at which spending function is to be evaluated or `NULL` if information-based spending is used).
  - `max_info` (when `timing` is `NULL`, this can be input as positive number to be used with `info` for information fraction at each analysis).

info
: Statistical information at all analyses, at least up to analysis k.

Value

A vector of the alpha spending per analysis.

Examples

```
# alpha-spending
par <- list(sf = gsDesign::sfLDOF, total_spend = 0.025)
gs_spending_combo(par, info = 1:3 / 3)

par <- list(sf = gsDesign::sfLDPocock, total_spend = 0.025)
gs_spending_combo(par, info = 1:3 / 3)

par <- list(sf = gsDesign::sfHSD, total_spend = 0.025, param = -40)
gs_spending_combo(par, info = 1:3 / 3)

# Kim-DeMets (power) Spending Function
par <- list(sf = gsDesign::sfPower, total_spend = 0.025, param = 1.5)
gs_spending_combo(par, info = 1:3 / 3)

# Exponential Spending Function
par <- list(sf = gsDesign::sfExponential, total_spend = 0.025, param = 1)
gs_spending_combo(par, info = 1:3 / 3)

# Two-parameter Spending Function Families
par <- list(sf = gsDesign::sfLogistic, total_spend = 0.025, param = c(.1, .4, .01, .1))
gs_spending_combo(par, info = 1:3 / 3)

par <- list(sf = gsDesign::sfBetaDist, total_spend = 0.025, param = c(.1, .4, .01, .1))
gs_spending_combo(par, info = 1:3 / 3)

par <- list(sf = gsDesign::sfCauchy, total_spend = 0.025, param = c(.1, .4, .01, .1))
gs_spending_combo(par, info = 1:3 / 3)

par <- list(sf = gsDesign::sfExtremeValue, total_spend = 0.025, param = c(.1, .4, .01, .1))
gs_spending_combo(par, info = 1:3 / 3)

par <- list(sf = gsDesign::sfExtremeValue2, total_spend = 0.025, param = c(.1, .4, .01, .1))
gs_spending_combo(par, info = 1:3 / 3)
```

```
par <- list(sf = gsDesign::sfNormal, total_spend = 0.025, param = c(.1, .4, .01, .1))
gs_spending_combo(par, info = 1:3 / 3)

# t-distribution Spending Function
par <- list(sf = gsDesign::sfTDist, total_spend = 0.025, param = c(-1, 1.5, 4))
gs_spending_combo(par, info = 1:3 / 3)

# Piecewise Linear and Step Function Spending Functions
par <- list(sf = gsDesign::sfLinear, total_spend = 0.025, param = c(.2, .4, .05, .2))
gs_spending_combo(par, info = 1:3 / 3)

par <- list(sf = gsDesign::sfStep, total_spend = 0.025, param = c(1 / 3, 2 / 3, .1, .1))
gs_spending_combo(par, info = 1:3 / 3)

# Pointwise Spending Function
par <- list(sf = gsDesign::sfPoints, total_spend = 0.025, param = c(.25, .25))
gs_spending_combo(par, info = 1:3 / 3)

# Truncated, trimmed and gapped spending functions
par <- list(sf = gsDesign::sfTruncated, total_spend = 0.025,
  param = list(trange = c(.2, .8), sf = gsDesign::sfHSD, param = 1))
gs_spending_combo(par, info = 1:3 / 3)

par <- list(sf = gsDesign::sfTrimmed, total_spend = 0.025,
  param = list(trange = c(.2, .8), sf = gsDesign::sfHSD, param = 1))
gs_spending_combo(par, info = 1:3 / 3)

par <- list(sf = gsDesign::sfGapped, total_spend = 0.025,
  param = list(trange = c(.2, .8), sf = gsDesign::sfHSD, param = 1))
gs_spending_combo(par, info = 1:3 / 3)

# Xi and Gallo conditional error spending functions
par <- list(sf = gsDesign::sfXG1, total_spend = 0.025, param = 0.5)
gs_spending_combo(par, info = 1:3 / 3)

par <- list(sf = gsDesign::sfXG2, total_spend = 0.025, param = 0.14)
gs_spending_combo(par, info = 1:3 / 3)

par <- list(sf = gsDesign::sfXG3, total_spend = 0.025, param = 0.013)
gs_spending_combo(par, info = 1:3 / 3)

# beta-spending
par <- list(sf = gsDesign::sfLDOF, total_spend = 0.2)
gs_spending_combo(par, info = 1:3 / 3)
```

---

gs_update_ahr — *Group sequential design using average hazard ratio under non-proportional hazards*

Description

Group sequential design using average hazard ratio under non-proportional hazards

Usage

```
gs_update_ahr(
  x = NULL,
  alpha = NULL,
  ustime = NULL,
  lstime = NULL,
  event_tbl = NULL
)
```

Arguments

`x`	A design created by either `gs_design_ahr()` or `gs_power_ahr()`.
`alpha`	Type I error for the updated design.
`ustime`	Default is NULL in which case upper bound spending time is determined by timing. Otherwise, this should be a vector of length k (total number of analyses) with the spending time at each analysis.
`lstime`	Default is NULL in which case lower bound spending time is determined by timing. Otherwise, this should be a vector of length k (total number of analyses) with the spending time at each analysis.
`event_tbl`	A data frame with two columns: (1) analysis and (2) event, which represents the events observed at each analysis per piecewise interval. This can be defined via the `pw_observed_event()` function or manually entered. For example, consider a scenario with two intervals in the piecewise model: the first interval lasts 6 months with a hazard ratio (HR) of 1, and the second interval follows with an HR of 0.6. The data frame `event_tbl = data.frame(analysis = c(1, 1, 2, 2), event = c(30, 100, 30, 200))` indicates that 30 events were observed during the delayed effect period, 130 events were observed at the IA, and 230 events were observed at the FA.

Value

A list with input parameters, enrollment rate, failure rate, analysis, and bound.

Examples

```
library(gsDesign)
library(gsDesign2)

alpha <- 0.025
beta <- 0.1
ratio <- 1

# Enrollment
enroll_rate <- define_enroll_rate(
```

```
  duration = c(2, 2, 10),
  rate = (1:3) / 3)

# Failure and dropout
fail_rate <- define_fail_rate(
  duration = c(3, Inf), fail_rate = log(2) / 9,
  hr = c(1, 0.6), dropout_rate = .0001)

# IA and FA analysis time
analysis_time <- c(20, 36)

# Randomization ratio
ratio <- 1

# ------------------------------------------------- #
# Two-sided asymmetric design,
# beta-spending with non-binding lower bound
# ------------------------------------------------- #
# Original design
x <- gs_design_ahr(
  enroll_rate = enroll_rate, fail_rate = fail_rate,
  alpha = alpha, beta = beta, ratio = ratio,
  info_scale = "h0_info",
  info_frac = NULL, analysis_time = c(20, 36),
  upper = gs_spending_bound,
  upar = list(sf = sfLDOF, total_spend = alpha),
  test_upper = TRUE,
  lower = gs_spending_bound,
  lpar = list(sf = sfLDOF, total_spend = beta),
  test_lower = c(TRUE, FALSE),
  binding = FALSE) |> to_integer()

planned_event_ia <- x$analysis$event[1]
planned_event_fa <- x$analysis$event[2]


# Updated design with 190 events observed at IA,
# where 50 events observed during the delayed effect.
# IA spending = observed events / final planned events, the remaining alpha will be allocated to FA.
gs_update_ahr(
  x = x,
  ustime = c(190 / planned_event_fa, 1),
  lstime = c(190 / planned_event_fa, 1),
  event_tbl = data.frame(analysis = c(1, 1),
                         event = c(50, 140)))

# Updated design with 190 events observed at IA, and 300 events observed at FA,
# where 50 events observed during the delayed effect.
# IA spending = observed events / final planned events, the remaining alpha will be allocated to FA.
gs_update_ahr(
  x = x,
  ustime = c(190 / planned_event_fa, 1),
  lstime = c(190 / planned_event_fa, 1),
```

```
  event_tbl = data.frame(analysis = c(1, 1, 2, 2),
                         event = c(50, 140, 50, 250)))

# Updated design with 190 events observed at IA, and 300 events observed at FA,
# where 50 events observed during the delayed effect.
# IA spending = minimal of planned and actual information fraction spending
gs_update_ahr(
  x = x,
  ustime = c(min(190, planned_event_ia) / planned_event_fa, 1),
  lstime = c(min(190, planned_event_ia) / planned_event_fa, 1),
  event_tbl = data.frame(analysis = c(1, 1, 2, 2),
                         event = c(50, 140, 50, 250)))

# Alpha is updated to 0.05
gs_update_ahr(x = x, alpha = 0.05)

# ------------------------------------------------- #
# Two-sided asymmetric stratified design,
# beta-spending with non-binding lower bound
# ------------------------------------------------- #
enroll_rate <- define_enroll_rate(stratum = c("A", "B"), duration = c(12, 12), rate = c(1, 1))

# We assumme there are 2 strata, "A" and "B".
# For each stratum, there are delayed effect for the first 3 months.
# After the delayed effect, the HR is 0.8 for stratum A and 0.5 for stratum B.
fail_rate <- define_fail_rate(stratum = c("A", "A", "B", "B"),
                              duration = c(3, Inf, 3, Inf),
                              fail_rate = log(2) / c(9, 9, 9, 15),
                              hr = c(1, 0.8, 1, 0.5),
                              dropout_rate = rep(0.001, 4))

# The original design assumes there are 2 IAs and 1 FA cutting by calendar time.
# The efficacy testing is conducted at IA2 and FA.
# The futility testing is conducted at IA1.
x <- gs_design_ahr(enroll_rate = enroll_rate,
                   fail_rate = fail_rate,
                   alpha = 0.0125,
                   beta = 0.1,
                   analysis = c(20, 28, 36),
                   upper = "gs_spending_bound",
                   upar = list(sf = "sfLDOF", total_spend = 0.0125),
                   lower = "gs_spending_bound",
                   lpar = list(sf = "sfHSD", total_spend = 0.1, param = -8),
                   test_upper = c(FALSE, TRUE, TRUE),
                   test_lower = c(TRUE, FALSE, FALSE)) |> to_integer()

# At time of analysis
# For IA1,
# - There are 70 events observed during the delayed effect period for stratum A.
# - There are 150 events observed after the delayed effect period for stratum A.
# - There are 75 events observed during the delayed effect period for stratum B.
# - There are 90 events observed after the delayed effect period for stratum B.
# For IA2,
```

```
# - There are 75 events observed during the delayed effect period for stratum A.
# - There are 210 events observed after the delayed effect period for stratum A.
# - There are 76 events observed during the delayed effect period for stratum B.
# - There are 136 events observed after the delayed effect period for stratum B.
# For FA,
# - There are 77 events observed during the delayed effect period for stratum A.
# - There are 245 events observed after the delayed effect period for stratum A.
# - There are 77 events observed during the delayed effect period for stratum B.
# - There are 170 events observed after the delayed effect period for stratum B.
event_tbl <- data.frame(analysis = c(1, 1, 1, 1,
                                     2, 2, 2, 2,
                                     3, 3, 3, 3),
                        stratum = c("A", "A", "B", "B", # IA1
                                    "A", "A", "B", "B", # IA2
                                    "A", "A", "B", "B"),# FA
                                  # event per interval per stratum at IA1
                        event = c(70, 150, 75, 90,
                                  # event per interval per stratum at IA2
                                  75, 210, 76, 136,
                                  # event per interval per stratum at FA
                                  77, 245, 77, 170))
observed_event <- (event_tbl |> dplyr::group_by(analysis) |> dplyr::summarize(x = sum(event)))$x

ustime <- pmin(x$analysis$event,
               observed_event) / x$analysis$event[3]
ustime[3] <- 1
lstime <- ustime

xu <- gs_update_ahr(x = x,
                    alpha = 0.015,
                    ustime = ustime,
                    lstime = lstime,
                    event_tbl = event_tbl
                    )
```

---

ppwe *Piecewise exponential cumulative distribution function*

Description

Computes the cumulative distribution function (CDF) or survival rate for a piecewise exponential distribution.

Usage

```
ppwe(x, duration, rate, lower_tail = FALSE)
```

Arguments

`x`	Times at which distribution is to be computed.
`duration`	A numeric vector of time duration.
`rate`	A numeric vector of event rate.
`lower_tail`	Indicator of whether lower (`TRUE`) or upper tail (`FALSE`; default) of CDF is to be computed.

Details

Suppose λ_i is the failure rate in the interval $(t_{i-1}, t_i], i = 1, 2, \ldots, M$ where $0 = t_0 < t_i \ldots, t_M = \infty$. The cumulative hazard function at an arbitrary time $t > 0$ is then:

$$\Lambda(t) = \sum_{i=1}^{M} \delta(t \leq t_i)(\min(t, t_i) - t_{i-1})\lambda_i.$$

The survival at time t is then

$$S(t) = \exp(-\Lambda(t)).$$

Value

A vector with cumulative distribution function or survival values.

Specification

- Validate if input enrollment rate is a strictly increasing non-negative numeric vector.
- Validate if input failure rate is of type data.frame.
- Validate if input failure rate contains duration column.
- Validate if input failure rate contains rate column.
- Validate if input lower_tail is logical.
- Convert rates to step function.
- Add times where rates change to enrollment rates.
- Make a tibble of the input time points x, duration, hazard rates at points, cumulative hazard and survival.
- Extract the expected cumulative or survival of piecewise exponential distribution.
- If input lower_tail is true, return the CDF, else return the survival for `ppwe`

Examples

```
# Plot a survival function with 2 different sets of time values
# to demonstrate plot precision corresponding to input parameters.

x1 <- seq(0, 10, 10 / pi)
duration <- c(3, 3, 1)
rate <- c(.2, .1, .005)

survival <- ppwe(
```

```
  x = x1,
  duration = duration,
  rate = rate
)
plot(x1, survival, type = "l", ylim = c(0, 1))

x2 <- seq(0, 10, .25)
survival <- ppwe(
  x = x2,
  duration = duration,
  rate = rate
)
lines(x2, survival, col = 2)
```

---

pw_info *Average hazard ratio under non-proportional hazards*

Description

Provides a geometric average hazard ratio under various non-proportional hazards assumptions for either single or multiple strata studies. The piecewise exponential distribution allows a simple method to specify a distribution and enrollment pattern where the enrollment, failure and dropout rates changes over time.

Usage

```
pw_info(
  enroll_rate = define_enroll_rate(duration = c(2, 2, 10), rate = c(3, 6, 9)),
  fail_rate = define_fail_rate(duration = c(3, 100), fail_rate = log(2)/c(9, 18), hr =
    c(0.9, 0.6), dropout_rate = 0.001),
  total_duration = 30,
  ratio = 1
)
```

Arguments

enroll_rate	An enroll_rate data frame with or without stratum created by define_enroll_rate().
fail_rate	A fail_rate data frame with or without stratum created by define_fail_rate().
total_duration	Total follow-up from start of enrollment to data cutoff; this can be a single value or a vector of positive numbers.
ratio	Experimental:Control randomization ratio.

Value

A data frame with time (from total_duration), stratum, t, hr (hazard ratio), event (expected number of events), info (information under given scenarios), info0 (information under related null hypothesis), and n (sample size) for each value of total_duration input

Examples

```
# Example: default
pw_info()

# Example: default with multiple analysis times (varying total_duration)
pw_info(total_duration = c(15, 30))

# Stratified population
enroll_rate <- define_enroll_rate(
  stratum = c(rep("Low", 2), rep("High", 3)),
  duration = c(2, 10, 4, 4, 8),
  rate = c(5, 10, 0, 3, 6)
)
fail_rate <- define_fail_rate(
  stratum = c(rep("Low", 2), rep("High", 2)),
  duration = c(1, Inf, 1, Inf),
  fail_rate = c(.1, .2, .3, .4),
  dropout_rate = .001,
  hr = c(.9, .75, .8, .6)
)
# Give results by change-points in the piecewise model
ahr(enroll_rate = enroll_rate, fail_rate = fail_rate, total_duration = c(15, 30))

# Same example, give results by strata and time period
pw_info(enroll_rate = enroll_rate, fail_rate = fail_rate, total_duration = c(15, 30))
```

---

s2pwe *Approximate survival distribution with piecewise exponential distribution*

Description

Converts a discrete set of points from an arbitrary survival distribution to a piecewise exponential approximation.

Usage

```
s2pwe(times, survival)
```

Arguments

times	Positive increasing times at which survival distribution is provided.
survival	Survival (1 - cumulative distribution function) at specified `times`.

Value

A tibble containing the duration and rate.

Specification

- Validate if input times is increasing positive finite numbers.
- Validate if input survival is numeric and same length as input times.
- Validate if input survival is positive, non-increasing, less than or equal to 1 and greater than 0.
- Create a tibble of inputs times and survival.
- Calculate the duration, hazard and the rate.
- Return the duration and rate by `s2pwe`

Examples

```
# Example: arbitrary numbers
s2pwe(1:9, (9:1) / 10)
# Example: lognormal
s2pwe(c(1:6, 9), plnorm(c(1:6, 9), meanlog = 0, sdlog = 2, lower.tail = FALSE))
```

`sequential_pval` *Sequential p-value computation for gsDesign2*

Description

`sequential_pval` computes a sequential p-value for a group sequential design created with `gs_design_ahr()` using a spending function approach. It is the minimum of repeated p-values computed at each analysis (Jennison and Turnbull, 2000). This is particularly useful for multiplicity methods such as the graphical method for group sequential designs where sequential p-values for multiple hypotheses can be used as nominal p-values to plug into a multiplicity graph. A sequential p-value is described as the minimum alpha level at which a one-sided group sequential bound would be rejected given interim and final observed results. Mild restrictions are required on spending functions used, but these are satisfied for commonly used spending functions such as the Lan-DeMets spending function approximating an O'Brien-Fleming bound or a Hwang-Shih-DeCani spending function.

Usage

```
sequential_pval(
  gs_design,
  event = NULL,
  z = NULL,
  ustime = NULL,
  interval = c(1e-05, 0.9999)
)
```

Arguments

`gs_design`	Group sequential design generated by `gs_design_ahr()`.
`event`	Event counts; numeric vector with increasing, positive values with at most one value greater than or equal to largest value in `gs_design$analysis$event`; NOTE: if NULL, planned `event` will be used (`gs_design$analysis$event`)

z	z-value tests corresponding to analyses in `info`; positive values indicate a positive finding; must have the same length as `info`.
ustime	Spending time for upper bound at specified analyses; specify default: `NULL` if this is to be based on information fraction; if not `NULL`, must have the same length as `info`; increasing positive values with at most 1 greater than or equal to 1.
interval	Interval for search to derive p-value; Default: `c(1e-05, 0.9999)`. Lower end of interval must be >0 and upper end must be < 1. The primary reason to not use the defaults would likely be if a test were vs a Type I error <0.0001.

Details

Solution is found with a search using `uniroot`. This finds the maximum alpha-level for which an efficacy bound is crossed, completely ignoring any futility bound.

Value

Sequential p-value (single numeric one-sided p-value between 0 and 1). Note that if the sequential p-value is less than the lower end of the input interval, the lower of interval will be returned. Similarly, if the sequential p-value is greater than the upper end of the input interval, then the upper end of interval is returned.

References

Jennison C and Turnbull BW (2000), *Group Sequential Methods with Applications to Clinical Trials*. Boca Raton: Chapman and Hall.

Liu, Qing, and Keaven M. Anderson. "On adaptive extensions of group sequential trials for clinical investigations." *Journal of the American Statistical Association* 103.484 (2008): 1621-1630.

Maurer, Willi, and Frank Bretz. "Multiple testing in group sequential trials using graphical approaches." *Statistics in Biopharmaceutical Research* 5.4 (2013): 311-320.

Examples

```
# Derive Group Sequential Design using gsDesign2
x <- gs_design_ahr(
  enroll_rate = define_enroll_rate(duration = c(2, 2, 2, 6),
                                   rate = c(2.5, 5, 7.5, 10)),
  fail_rate = define_fail_rate(duration = Inf, fail_rate = log(2) / 6,
                               hr = 0.6, dropout_rate = .001),
  info_frac = c(.5, .65, .8, 1),
  analysis_time = 30,
  upper = gs_spending_bound,
  upar = list(sf = "sfLDOF", total_spend = 0.025),
  lower = gs_spending_bound,
  lpar = list(sf = "sfHSD", total_spend = 0.1, param = 2),
  binding = FALSE
)

# Analysis at interim analysis 2
sequential_pval(gs_design = x, event = c(100, 160), z = c(1.5, 2))
```

```
# Use planned spending for interim and final analyses
sequential_pval(gs_design = x,
                event = c(100, 160, 190),
                z = c(1.5, 2, 2.5))
```

---

summary.fixed_design *Summary for fixed design or group sequential design objects*

Description

Summary for fixed design or group sequential design objects

Usage

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

## S3 method for class 'gs_design'
summary(
  object,
  analysis_vars = NULL,
  analysis_decimals = NULL,
  col_vars = NULL,
  col_decimals = NULL,
  bound_names = c("Efficacy", "Futility"),
  display_spending_time = FALSE,
  ...
)
```

Arguments

`object`	A design object returned by fixed_design_xxx() and gs_design_xxx().
`...`	Additional parameters (not used).
`analysis_vars`	The variables to be put at the summary header of each analysis.
`analysis_decimals`	The displayed number of digits of `analysis_vars`. If the vector is unnamed, it must match the length of `analysis_vars`. If the vector is named, you only have to specify the number of digits for the variables you want to be displayed differently than the defaults.
`col_vars`	The variables to be displayed.
`col_decimals`	The decimals to be displayed for the displayed variables in `col_vars`. If the vector is unnamed, it must match the length of `col_vars`. If the vector is named, you only have to specify the number of digits for the columns you want to be displayed differently than the defaults.
`bound_names`	Names for bounds; default is `c("Efficacy", "Futility")`.

`display_spending_time`
: A logical value (TRUE/FALSE) indicating if the spending time is summarized in the table.

Value

A summary table (data frame).

Examples

```
# Enrollment rate
enroll_rate <- define_enroll_rate(
  duration = 18,
  rate = 20
)

# Failure rates
fail_rate <- define_fail_rate(
  duration = c(4, 100),
  fail_rate = log(2) / 12,
  hr = c(1, .6),
  dropout_rate = .001
)

# Study duration in months
study_duration <- 36

# Experimental / Control randomization ratio
ratio <- 1

# 1-sided Type I error
alpha <- 0.025
# Type II error (1 - power)
beta <- 0.1

# AHR ----
# under fixed power
fixed_design_ahr(
  alpha = alpha,
  power = 1 - beta,
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  study_duration = study_duration,
  ratio = ratio
) |> summary()

# FH ----
# under fixed power
fixed_design_fh(
  alpha = alpha,
  power = 1 - beta,
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
```

```
  study_duration = study_duration,
  ratio = ratio
) |> summary()

# Design parameters ----
library(gsDesign)
library(gsDesign2)

# enrollment/failure rates
enroll_rate <- define_enroll_rate(
  stratum = "All",
  duration = 12,
  rate = 1
)
fail_rate <- define_fail_rate(
  duration = c(4, 100),
  fail_rate = log(2) / 12,
  hr = c(1, .6),
  dropout_rate = .001
)

# Information fraction
info_frac <- (1:3) / 3

# Analysis times in months; first 2 will be ignored as info_frac will not be achieved
analysis_time <- c(.01, .02, 36)

# Experimental / Control randomization ratio
ratio <- 1

# 1-sided Type I error
alpha <- 0.025

# Type II error (1 - power)
beta <- .1

# Upper bound
upper <- gs_spending_bound
upar <- list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL)

# Lower bound
lower <- gs_spending_bound
lpar <- list(sf = gsDesign::sfHSD, total_spend = 0.1, param = 0, timing = NULL)

# test in COMBO
fh_test <- rbind(
 data.frame(rho = 0, gamma = 0, tau = -1, test = 1, analysis = 1:3, analysis_time = c(12, 24, 36)),
 data.frame(rho = c(0, 0.5), gamma = 0.5, tau = -1, test = 2:3, analysis = 3, analysis_time = 36)
)

# Example 1 ----

x_ahr <- gs_design_ahr(
```

```
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  info_frac = info_frac, # Information fraction
  analysis_time = analysis_time,
  ratio = ratio,
  alpha = alpha,
  beta = beta,
  upper = upper,
  upar = upar,
  lower = lower,
  lpar = lpar
)

x_ahr |> summary()

# Customize the digits to display
x_ahr |> summary(analysis_vars = c("time", "event", "info_frac"), analysis_decimals = c(1, 0, 2))

# Customize the labels of the crossing probability
x_ahr |> summary(bound_names = c("A is better", "B is better"))

# Customize the variables to be summarized for each analysis
x_ahr |> summary(analysis_vars = c("n", "event"), analysis_decimals = c(1, 1))

# Customize the digits for the columns
x_ahr |> summary(col_decimals = c(z = 4))

# Customize the columns to display
x_ahr |> summary(col_vars = c("z", "~hr at bound", "nominal p"))

# Customize columns and digits
x_ahr |> summary(col_vars = c("z", "~hr at bound", "nominal p"),
                 col_decimals = c(4, 2, 2))


# Example 2 ----

x_wlr <- gs_design_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  weight = list(method = "fh", param = list(rho = 0, gamma = 0.5)),
  info_frac = NULL,
  analysis_time = sort(unique(x_ahr$analysis$time)),
  ratio = ratio,
  alpha = alpha,
  beta = beta,
  upper = upper,
  upar = upar,
  lower = lower,
  lpar = lpar
)
x_wlr |> summary()
```

```
# Maxcombo ----

x_combo <- gs_design_combo(
  ratio = 1,
  alpha = 0.025,
  beta = 0.2,
  enroll_rate = define_enroll_rate(duration = 12, rate = 500 / 12),
  fail_rate = tibble::tibble(
    stratum = "All",
    duration = c(4, 100),
    fail_rate = log(2) / 15, hr = c(1, .6), dropout_rate = .001
  ),
  fh_test = fh_test,
  upper = gs_spending_combo,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025),
  lower = gs_spending_combo,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.2)
)
x_combo |> summary()

# Risk difference ----

gs_design_rd(
  p_c = tibble::tibble(stratum = "All", rate = .2),
  p_e = tibble::tibble(stratum = "All", rate = .15),
  info_frac = c(0.7, 1),
  rd0 = 0,
  alpha = .025,
  beta = .1,
  ratio = 1,
  stratum_prev = NULL,
  weight = "unstratified",
  upper = gs_b,
  lower = gs_b,
  upar = gsDesign::gsDesign(
    k = 3, test.type = 1, sfu = gsDesign::sfLDOF, sfupar = NULL
  )$upper$bound,
  lpar = c(qnorm(.1), rep(-Inf, 2))
) |> summary()
```

`text_summary` *Generates a textual summary of a group sequential design using the AHR method.*

Description

Generates a textual summary of a group sequential design using the AHR method.

Usage

```
text_summary(x, information = FALSE, time_unit = "months")
```

Arguments

x	A design object created by gs_design_ahr() with or without to_integer().
information	A logical value indicating whether to include statistical information in the textual summary. Default is FALSE.
time_unit	A character string specifying the time unit used in the design. Options include "days", "weeks", "months" (default), and "years".

Value

A character string containing a paragraph that summarizes the design.

Examples

```
library(gsDesign)

# Text summary of a 1-sided design
x <- gs_design_ahr(info_frac = 1:3/3, test_lower = FALSE) |> to_integer()
x |> text_summary()

# Text summary of a 2-sided symmetric design
x <- gs_design_ahr(info_frac = 1:3/3,
                   upper = gs_spending_bound, lower = gs_spending_bound,
                   upar = list(sf = sfLDOF, total_spend = 0.025),
                   lpar = list(sf = sfLDOF, total_spend = 0.025),
                   binding = TRUE, h1_spending = FALSE) |> to_integer()
x |> text_summary()

# Text summary of a asymmetric 2-sided design with beta-spending and non-binding futility bound
x <- gs_design_ahr(info_frac = 1:3/3, alpha = 0.025, beta = 0.1,
                   upper = gs_spending_bound, lower = gs_spending_bound,
                   upar = list(sf = sfLDOF, total_spend = 0.025),
                   lpar = list(sf = sfHSD, total_spend = 0.1, param = -4),
                   binding = FALSE, h1_spending = TRUE) |> to_integer()
x |> text_summary()

# Text summary of a asymmetric 2-sided design with fixed non-binding futility bound
x <- gs_design_ahr(info_frac = 1:3/3, alpha = 0.025, beta = 0.1,
                   upper = gs_spending_bound, lower = gs_b,
                   upar = list(sf = sfLDOF, total_spend = 0.025),
                   test_upper = c(FALSE, TRUE, TRUE),
                   lpar = c(-1, -Inf, -Inf),
                   test_lower = c(TRUE, FALSE, FALSE),
                   binding = FALSE, h1_spending = TRUE) |> to_integer()
x |> text_summary()

# If there are >5 pieces of HRs, we provide a brief summary of HR.
gs_design_ahr(
  fail_rate = define_fail_rate(duration = c(rep(3, 5), Inf),
                               hr = c(0.9, 0.8, 0.7, 0.6, 0.5, 0.4),
                               fail_rate = log(2) / 10, dropout_rate = 0.001),
  info_frac = 1:3/3, test_lower = FALSE) |>
```

```
  text_summary()

# Text summary of a fixed design created with gs_design_ahr()
x <- gs_design_ahr(
  upper = gs_b,
  lower = gs_b,
  upar = qnorm(1 - 0.025),
  lpar = -Inf
)
x |> text_summary()
```

to_integer *Round sample size and events*

Description

Round sample size and events

Usage

```
to_integer(x, ...)

## S3 method for class 'fixed_design'
to_integer(x, round_up_final = TRUE, ratio = x$input$ratio, ...)

## S3 method for class 'gs_design'
to_integer(x, round_up_final = TRUE, ratio = x$input$ratio, ...)
```

Arguments

x	An object returned by fixed_design_xxx() and gs_design_xxx().
...	Additional parameters (not used).
round_up_final	Events at final analysis is rounded up if TRUE; otherwise, just rounded, unless it is very close to an integer.
ratio	Positive integer for randomization ratio (experimental:control). A positive integer will result in rounded sample size, which is a multiple of (ratio + 1). A positive non-integer will result in round sample size, which may not be a multiple of (ratio + 1). A negative number will result in an error.

Details

For the sample size of the fixed design:

- When `ratio` is a positive integer, the sample size is rounded up to a multiple of `ratio + 1` if `round_up_final = TRUE`, and just rounded to a multiple of `ratio + 1` if `round_up_final = FALSE`.

- When `ratio` is a positive non-integer, the sample size is rounded up if `round_up_final = TRUE`, (may not be a multiple of `ratio + 1`), and just rounded if `round_up_final = FALSE` (may not be a multiple of `ratio + 1`). Note the default `ratio` is taken from `x$input$ratio`.

For the number of events of the fixed design:

- If the continuous event is very close to an integer within 0.01 differences, say 100.001 or 99.999, then the integer events is 100.
- Otherwise, round up if `round_up_final = TRUE` and round if `round_up_final = FALSE`.

For the sample size of group sequential designs:

- When `ratio` is a positive integer, the final sample size is rounded to a multiple of `ratio + 1`.
  - For 1:1 randomization (experimental:control), set `ratio = 1` to round to an even sample size.
  - For 2:1 randomization, set `ratio = 2` to round to a multiple of 3.
  - For 3:2 randomization, set `ratio = 4` to round to a multiple of 5.
  - Note that for the final analysis, the sample size is rounded up to the nearest multiple of `ratio + 1` if `round_up_final = TRUE`. If `round_up_final = FALSE`, the final sample size is rounded to the nearest multiple of `ratio + 1`.
- When `ratio` is positive non-integer, the final sample size MAY NOT be rounded to a multiple of `ratio + 1`.
  - The final sample size is rounded up if `round_up_final = TRUE`.
  - Otherwise, it is just rounded.

For the events of group sequential designs:

- For events at interim analysis, it is rounded.
- For events at final analysis:
  - If the continuous event is very close to an integer within 0.01 differences, say 100.001 or 99.999, then the integer events is 100.
  - Otherwise, final events is rounded up if `round_up_final = TRUE` and rounded if `round_up_final = FALSE`.

Value

A list similar to the output of fixed_design_xxx() and gs_design_xxx(), except the sample size is an integer.

Examples

```
library(gsDesign2)

# Average hazard ratio

x <- fixed_design_ahr(
  alpha = .025, power = .9,
  enroll_rate = define_enroll_rate(duration = 18, rate = 1),
  fail_rate = define_fail_rate(
```

```
    duration = c(4, 100),
    fail_rate = log(2) / 12, hr = c(1, .6),
    dropout_rate = .001
  ),
  study_duration = 36
)
x |>
  to_integer() |>
  summary()

# FH
x <- fixed_design_fh(
  alpha = 0.025, power = 0.9,
  enroll_rate = define_enroll_rate(duration = 18, rate = 20),
  fail_rate = define_fail_rate(
    duration = c(4, 100),
    fail_rate = log(2) / 12,
    hr = c(1, .6),
    dropout_rate = .001
  ),
  rho = 0.5, gamma = 0.5,
  study_duration = 36, ratio = 1
)
x |>
  to_integer() |>
  summary()

# MB
x <- fixed_design_mb(
  alpha = 0.025, power = 0.9,
  enroll_rate = define_enroll_rate(duration = 18, rate = 20),
  fail_rate = define_fail_rate(
    duration = c(4, 100),
    fail_rate = log(2) / 12, hr = c(1, .6),
    dropout_rate = .001
  ),
  tau = Inf, w_max = 2,
  study_duration = 36, ratio = 1
)
x |>
  to_integer() |>
  summary()


# Example 1: Information fraction based spending
gs_design_ahr(
  analysis_time = c(18, 30),
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL),
  lower = gs_b,
  lpar = c(-Inf, -Inf)
) |>
  to_integer() |>
```

```
  summary()

gs_design_wlr(
  analysis_time = c(18, 30),
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL),
  lower = gs_b,
  lpar = c(-Inf, -Inf)
) |>
  to_integer() |>
  summary()

gs_design_rd(
  p_c = tibble::tibble(stratum = c("A", "B"), rate = c(.2, .3)),
  p_e = tibble::tibble(stratum = c("A", "B"), rate = c(.15, .27)),
  weight = "ss",
  stratum_prev = tibble::tibble(stratum = c("A", "B"), prevalence = c(.4, .6)),
  info_frac = c(0.7, 1),
  upper = gs_spending_bound,
  upar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL),
  lower = gs_b,
  lpar = c(-Inf, -Inf)
) |>
  to_integer() |>
  summary()

# Example 2: Calendar based spending
x <- gs_design_ahr(
  upper = gs_spending_bound,
  analysis_time = c(18, 30),
  upar = list(
    sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL,
    timing = c(18, 30) / 30
  ),
  lower = gs_b,
  lpar = c(-Inf, -Inf)
) |> to_integer()

# The IA nominal p-value is the same as the IA alpha spending
x$bound$`nominal p`[1]
gsDesign::sfLDOF(alpha = 0.025, t = 18 / 30)$spend
```

`wlr_weight` *Weight functions for weighted log-rank test*

Description

- `wlr_weight_fh` is Fleming-Harrington, FH(rho, gamma) weight function.
- `wlr_weight_1` is constant for log rank test.

- `wlr_weight_power` is Gehan-Breslow and Tarone-Ware weight function.
- `wlr_weight_mb` is Magirr (2021) weight function.

Usage

```
wlr_weight_fh(x, arm0, arm1, rho = 0, gamma = 0, tau = NULL)

wlr_weight_1(x, arm0, arm1)

wlr_weight_n(x, arm0, arm1, power = 1)

wlr_weight_mb(x, arm0, arm1, tau = NULL, w_max = Inf)
```

Arguments

`x`	A vector of numeric values.
`arm0`	An `arm` object defined in the npsurvSS package.
`arm1`	An `arm` object defined in the npsurvSS package.
`rho`	A scalar parameter that controls the type of test.
`gamma`	A scalar parameter that controls the type of test.
`tau`	A scalar parameter of the cut-off time for modest weighted log rank test.
`power`	A scalar parameter that controls the power of the weight function.
`w_max`	A scalar parameter of the cut-off weight for modest weighted log rank test.

Value

A vector of weights.

A vector of weights.

A vector of weights.

A vector of weights.

Specification

- Compute the sample size via the sum of arm sizes.
- Compute the proportion of size in the two arms.
- If the input tau is specified, define time up to the cut off time tau.
- Compute the CDF using the proportion of the size in the two arms and `npsruvSS::psurv()`.
- Return the Fleming-Harrington weights for weighted Log-rank test.

Examples

```
enroll_rate <- define_enroll_rate(
  duration = c(2, 2, 10),
  rate = c(3, 6, 9)
)

fail_rate <- define_fail_rate(
  duration = c(3, 100),
  fail_rate = log(2) / c(9, 18),
  hr = c(.9, .6),
  dropout_rate = .001
)

gs_arm <- gs_create_arm(enroll_rate, fail_rate, ratio = 1)
arm0 <- gs_arm$arm0
arm1 <- gs_arm$arm1

wlr_weight_fh(1:3, arm0, arm1, rho = 0, gamma = 0, tau = NULL)
enroll_rate <- define_enroll_rate(
  duration = c(2, 2, 10),
  rate = c(3, 6, 9)
)

fail_rate <- define_fail_rate(
  duration = c(3, 100),
  fail_rate = log(2) / c(9, 18),
  hr = c(.9, .6),
  dropout_rate = .001
)

gs_arm <- gs_create_arm(enroll_rate, fail_rate, ratio = 1)
arm0 <- gs_arm$arm0
arm1 <- gs_arm$arm1

wlr_weight_1(1:3, arm0, arm1)
enroll_rate <- define_enroll_rate(
  duration = c(2, 2, 10),
  rate = c(3, 6, 9)
)

fail_rate <- define_fail_rate(
  duration = c(3, 100),
  fail_rate = log(2) / c(9, 18),
  hr = c(.9, .6),
  dropout_rate = .001
)

gs_arm <- gs_create_arm(enroll_rate, fail_rate, ratio = 1)
arm0 <- gs_arm$arm0
arm1 <- gs_arm$arm1

wlr_weight_n(1:3, arm0, arm1, power = 2)
```

```
enroll_rate <- define_enroll_rate(
  duration = c(2, 2, 10),
  rate = c(3, 6, 9)
)

fail_rate <- define_fail_rate(
  duration = c(3, 100),
  fail_rate = log(2) / c(9, 18),
  hr = c(.9, .6),
  dropout_rate = .001
)

gs_arm <- gs_create_arm(enroll_rate, fail_rate, ratio = 1)
arm0 <- gs_arm$arm0
arm1 <- gs_arm$arm1

wlr_weight_mb(1:3, arm0, arm1, tau = -1, w_max = 1.2)
```

Index