

# Package ‘gsDesign2’

August 23, 2023

**Title** Group Sequential Design with Non-Constant Effect

**Version** 1.1.0

**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 (>= 3.5.0)

**Imports** corpcor, dplyr, gsDesign, gt, methods, mvtnorm, npsurvSS,  
rlang, tibble, tidyr, utils, Rcpp

**Suggests** covr, ggplot2, kableExtra, knitr, rmarkdown, testthat

**VignetteBuilder** knitr

**LinkingTo** Rcpp

**RoxygenNote** 7.2.3

**NeedsCompilation** yes

**Author** Keaven Anderson [aut],  
Yilong Zhang [aut],  
Yujie Zhao [aut, cre],  
Jianxiao Yang [aut],  
Nan Xiao [aut],  
Amin Shirazi [ctb],  
Ruixue Wang [ctb],

Yi Cui [ctb],  
 Ping Yang [ctb],  
 Xin Tong Li [ctb],  
 Chenxiang Li [ctb],  
 Hongtao Zhang [ctb],  
 Yalin Zhu [ctb],  
 John Blischak [ctb],  
 Merck & Co., Inc., Rahway, NJ, USA and its affiliates [cph]

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

**Repository** CRAN

**Date/Publication** 2023-08-23 15:40:02 UTC

## R topics documented:

ahr . . . . .	3
as_gt . . . . .	5
define_enroll_rate . . . . .	9
define_fail_rate . . . . .	9
expected_accrual . . . . .	10
expected_event . . . . .	12
expected_time . . . . .	14
fixed_design_ahr . . . . .	15
fixed_design_fh . . . . .	17
fixed_design_lf . . . . .	18
fixed_design_maxcombo . . . . .	20
fixed_design_mb . . . . .	21
fixed_design_milestone . . . . .	23
fixed_design_rd . . . . .	24
fixed_design_rmst . . . . .	25
gs_b . . . . .	27
gs_create_arm . . . . .	28
gs_design_ahr . . . . .	29
gs_design_combo . . . . .	33
gs_design_npe . . . . .	36
gs_design_rd . . . . .	41
gs_design_wlr . . . . .	43
gs_info_ahr . . . . .	47
gs_info_combo . . . . .	49
gs_info_rd . . . . .	50
gs_info_wlr . . . . .	53
gs_power_ahr . . . . .	55
gs_power_combo . . . . .	57
gs_power_npe . . . . .	59
gs_power_rd . . . . .	63
gs_power_wlr . . . . .	68
gs_spending_bound . . . . .	73
gs_spending_combo . . . . .	75

ppwe . . . . .	76
pw_info . . . . .	77
s2pwe . . . . .	79
summary.fixed_design . . . . .	80
to_integer . . . . .	84
wlr_weight . . . . .	86

<b>Index</b>	<b>89</b>
--------------	-----------

---

ahr	<i>Average hazard ratio under non-proportional hazards (test version)</i>
-----	---

---

## 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 <a href="#">define_enroll_rate()</a> .
fail_rate	A fail_rate data frame with or without stratum created by <a href="#">define_fail_rate()</a> .
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	Ratio of experimental to control randomization.

## Value

A tibble with time (from total\_duration), ahr (average hazard ratio), 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 tibble 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: default
ahr()

# Example: default with multiple analysis times (varying total_duration)
ahr(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 = 1,
  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))

```

---

as_gt	<i>Convert summary table of a fixed or group sequential design object to a gt object</i>
-------	--

---

### 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'
as_gt(x, title = NULL, footnote = NULL, ...)

## S3 method for class 'gs_design'
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 = TRUE,
  full_alpha = 0.025,
  ...
)

```

### 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.

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.
full_alpha	The full alpha used in the design, the default is 0.025. when the futility bound is non-binding, a footnote will be displayed, saying the smaller value subtracts the probability of crossing a futility bound before crossing an efficacy bound at a later analysis under the null hypothesis.

### Value

A gt\_tbl object.

### Examples

```
library(dplyr)
library(tibble)

# 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
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()

# ----- #
#           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_gt()

# the default output
library(dplyr)

gs_design_ahr() %>%
  summary() %>%
  as_gt()

gs_power_ahr() %>%
  summary() %>%
  as_gt()

gs_design_wlr() %>%
  summary() %>%
  as_gt()

gs_power_wlr() %>%
  summary() %>%
  as_gt()

gs_power_combo() %>%
  summary() %>%
  as_gt()

gs_design_rd() %>%
  summary() %>%
  as_gt()
```

```

gs_power_rd() %>%
  summary() %>%
  as_gt()

# usage of title = ..., subtitle = ...
# to edit the title/subtitle
gs_power_wlr() %>%
  summary() %>%
  as_gt(
    title = "Bound Summary",
    subtitle = "from gs_power_wlr"
  )

# usage of colname_spanner = ..., colname_spannersub = ...
# to edit the spanner and its sub-spanner
gs_power_wlr() %>%
  summary() %>%
  as_gt(
    colname_spanner = "Cumulative probability to cross boundaries",
    colname_spannersub = c("under H1", "under H0")
  )

# usage of footnote = ...
# to edit the footnote
gs_power_wlr() %>%
  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")
    )
  )

# usage of display_bound = ...
# to either show efficacy bound or futility bound, or both(default)
gs_power_wlr() %>%
  summary() %>%
  as_gt(display_bound = "Efficacy")

# usage of display_columns = ...
# to select the columns to display in the summary table
gs_power_wlr() %>%
  summary() %>%
  as_gt(display_columns = c("Analysis", "Bound", "Nominal p", "Z", "Probability"))

```

---

define\_enroll\_rate      *Define enrollment rate*

---

**Description**

Define enrollment rate

**Usage**

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

**Arguments**

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

**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(
  stratum = c("low", "low", "high"),
  duration = c(2, 2, 10),
  rate = c(3, 6, 9)
)
```

---

define\_fail\_rate      *Define fail rate*

---

**Description**

Define fail rate

**Usage**

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

**Arguments**

duration	A numeric vector of piecewise study duration interval.
fail_rate	A numeric vector of failure rate in each duration.
dropout_rate	A numeric vector of dropout rate in each duration.
hr	A numeric vector of hazard ratio.
stratum	A character vector of stratum name.

**Value**

A fail\_rate data frame.

**Examples**

```
# Define enroll rate without stratum
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(
  duration = c(3, 100),
  fail_rate = log(2) / c(9, 18),
  hr = c(.9, .6),
  dropout_rate = .001,
  stratum = c("low", "high")
)
```

---

expected_accrual	<i>Piecewise constant expected accrual</i>
------------------	--

---

**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)
  )
)

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)
  )
)

```

---

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 <a href="#">define_enroll_rate()</a> .
fail_rate	A fail_rate data frame with or without stratum created by <a href="#">define_fail_rate()</a> .
total_duration	Total follow-up from start of enrollment to data cutoff.
simple	If default (TRUE), return numeric expected number of events, otherwise a tibble 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 tibble 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 tibble correspond to the input tibble `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 tibble 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 tibble 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 tibble 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
)

```

---

expected_time	<i>Predict time at which a targeted event count is achieved</i>
---------------	---

---

### 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 <a href="#">define_enroll_rate()</a> .
fail_rate	A fail_rate data frame with or without stratum created by <a href="#">define_fail_rate()</a> .
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 tibble 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 tibble 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
)
```

**Description**

Computes fixed design sample size (given power) or power (given sample size) for AHR method. Returns a list with a basic summary.

**Usage**

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

**Arguments**

enroll_rate	Enrollment rates.
fail_rate	Failure and dropout rates.
alpha	One-sided Type I error.
power	Power (NULL to compute power or strictly between 0 and 1 - alpha otherwise).
ratio	Experimental:Control randomization ratio (not yet implemented).
study_duration	Study duration.
event	Targeted event at each analysis.

**Value**

A table.

**Examples**

```
library(dplyr)

# 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()
```

---

 fixed\_design\_fh

*Fixed design using Fleming-Harrington method*


---

## Description

Computes fixed design sample size (given power) or power (given sample size) for Fleming-Harrington method (Farrington and Manning, 1990). Returns a list with a basic summary.

## Usage

```
fixed_design_fh(
  alpha = 0.025,
  power = NULL,
  ratio = 1,
  study_duration = 36,
  enroll_rate,
  fail_rate,
  rho = 0,
  gamma = 0
)
```

## Arguments

alpha	One-sided Type I error.
power	Power (NULL to compute power or strictly between 0 and 1 - alpha otherwise).
ratio	Experimental:Control randomization ratio (not yet implemented).
study_duration	Study duration.
enroll_rate	Enrollment rates.
fail_rate	Failure and dropout rates.
rho	test parameter in Fleming-Harrington method.
gamma	test parameter in Fleming-Harrington method.

**Value**

A table.

**Examples**

```
library(dplyr)

# 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()
```

---

fixed\_design\_lf

*Fixed design using Lachin-Foulkes method*

---

**Description**

Computes fixed design sample size (given power) or power (given sample size) for Lachin-Foulkes method (Lachin and Foulkes, 1986). Returns a list with a basic summary.

**Usage**

```
fixed_design_lf(
  alpha = 0.025,
```

```

    power = NULL,
    ratio = 1,
    study_duration = 36,
    enroll_rate,
    fail_rate
  )

```

### Arguments

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.
enroll_rate	Enrollment rates.
fail_rate	Failure and dropout rates.

### Value

A table.

### Examples

```

library(dplyr)

# 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_fh(
  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()

```

---

fixed\_design\_maxcombo *Fixed design using MaxCombo method*

---

### Description

Computes fixed design sample size (given power) or power (given sample size) for MaxCombo method. Returns a list with a basic summary.

### Usage

```
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)  
)
```

### Arguments

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.
enroll_rate	Enrollment rates.
fail_rate	Failure and dropout rates.
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	A vector of numbers paring with gamma and rho for maxcombo test.

### Value

A table.

### Examples

```
library(dplyr)  
  
# 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()

```

---

 fixed\_design\_mb

*Fixed design using Magirr-Burman method*


---

## Description

Computes fixed design sample size (given power) or power (given sample size) for Magirr-Burman method. Returns a list with a basic summary.

## Usage

```

fixed_design_mb(
  alpha = 0.025,
  power = NULL,
  ratio = 1,
  study_duration = 36,
  enroll_rate,
  fail_rate,
  tau = 6
)

```

**Arguments**

alpha	One-sided Type I error.
power	Power (NULL to compute power or strictly between 0 and 1 - alpha otherwise).
ratio	Experimental:Control randomization ratio.
study_duration	Study duration.
enroll_rate	Enrollment rates.
fail_rate	Failure and dropout rates.
tau	Test parameter of Magirr-Burman method.

**Value**

A table.

**Examples**

```
library(dplyr)

# 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
)
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
)
x %>% summary()
```

---

`fixed_design_milestone`*Fixed design using milestone method*

---

**Description**

Computes fixed design sample size (given power) or power (given sample size) for milestone method (Yung and Liu, 2020). Returns a list with a basic summary.

**Usage**

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

**Arguments**

<code>alpha</code>	One-sided Type I error (strictly between 0 and 1).
<code>power</code>	Power (NULL to compute power or strictly between 0 and 1 - alpha otherwise).
<code>ratio</code>	Experimental:Control randomization ratio.
<code>enroll_rate</code>	Enrollment rates.
<code>fail_rate</code>	Failure and dropout rates.
<code>study_duration</code>	Study duration.
<code>tau</code>	Test parameter of milestone method.

**Value**

A table.

**Examples**

```
library(dplyr)  
  
# 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()

```

---

 fixed\_design\_rd

*Fixed design for binary outcome measuring in risk difference*


---

## Description

Computes fixed design sample size (given power) or power (given sample size) for binary outcome measuring in risk difference (Mehrotra and Railkar, 2000). Returns a list with a basic summary.

## Usage

```

fixed_design_rd(
  alpha = 0.025,
  power = NULL,
  ratio = 1,
  p_c,
  p_e,
  rd0 = 0,
  n = NULL
)

```

## Arguments

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.

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 table.

**Examples**

```
library(dplyr)

# 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()
```

---

fixed_design_rmst	<i>Fixed design using RMST method</i>
-------------------	---------------------------------------

---

**Description**

Computes fixed design sample size (given power) or power (given sample size) for RMST methods (Yung and Liu, 2020). Returns a list with a basic summary.

**Usage**

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

**Arguments**

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.
enroll_rate	Enrollment rates.
fail_rate	Failure and dropout rates.
tau	Test parameter in RMST.

**Value**

A table.

**Examples**

```
library(dplyr)

# 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\_prob(), gs\_power\_nph() and gs\_design\_nph(); it simply returns the vector input in the input vector Z or, if k is specified, par[k]j 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.

### 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)

```

---

<code>gs_create_arm</code>	<i>Create npsurvSS arm object</i>
----------------------------	-----------------------------------

---

## Description

Create npsurvSS arm object

## Usage

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

## Arguments

<code>enroll_rate</code>	Enrollment rates.
<code>fail_rate</code>	Failure and dropout rates.
<code>ratio</code>	Experimental:Control randomization ratio.
<code>total_time</code>	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	<i>Group sequential design using average hazard ratio under non-proportional hazards</i>
---------------	--

---

**Description**

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_b,
  upar = gsDesign::gsDesign(k = 3, test.type = 1, n.I = c(0.25, 0.75, 1), sfu = sfLDOF,
    sfupar = NULL)$upper$bound,
  lower = gs_b,
  lpar = c(qnorm(0.1), -Inf, -Inf),
  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, 100)
)

```

**Arguments**

enroll_rate	Enrollment rates.
fail_rate	Failure and dropout rates.
alpha	One-sided Type I error.
beta	Type II error.
info_frac	Targeted information fraction at each analysis.
analysis_time	Minimum time of analysis.
ratio	Experimental:Control randomization ratio (not yet implemented).
binding	Indicator of whether futility bound is binding; default of FALSE is recommended.
upper	Function to compute upper bound.
upar	Parameters passed to upper.
lower	Function to compute lower bound.
lpar	Parameters passed to lower.
h1_spending	Indicator that lower bound to be set by spending under alternate hypothesis (input fail_rate) if spending is used for 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.
test_lower	Indicator of which analyses should include an 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.
info_scale	Information scale for calculation. Options are: <ul style="list-style-type: none"> <li>• "h0_h1_info" (default): variance under both null and alternative hypotheses is used.</li> <li>• "h0_info": variance under null hypothesis is used.</li> <li>• "h1_info": variance under alternative hypothesis is used.</li> </ul>
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).
interval	An interval that is presumed to include the time at which expected event count is equal to targeted event.

**Details**

To be added.

**Value**

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

## 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)
library(dplyr)

# ----- #
#   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	<i>Group sequential design using MaxCombo test under non-proportional hazards</i>
-----------------	---

---

### 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	Enrollment rates.
fail_rate	Failure and dropout rates.
fh_test	A data frame to summarize the test in each analysis. See examples for its data structure.

ratio	Experimental:Control randomization ratio (not yet implemented).
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.
upar	Parameters passed to upper.
lower	Function to compute lower bound.
lpar	Parameters passed to lower.
algorithm	an object of class <a href="#">GenzBretz</a> , <a href="#">Miwa</a> or <a href="#">TVPACK</a> 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 <a href="#">mvtnorm::pmvnorm</a> .

### Value

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

### Examples

```
# The example is slow to run
library(dplyr)
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	<i>Group sequential design computation with non-constant effect and information</i>
---------------	---

---

## Description

Derives group sequential design size, bounds and boundary crossing probabilities based on proportionate information and effect size at analyses. It allows a non-constant treatment effect over time, but also can be applied for the usual homogeneous effect size designs. It requires treatment effect and proportionate statistical information at each analysis as well as a method of deriving bounds, such as spending. The routine enables two things not available in the gsDesign package:

1. non-constant effect, 2) more flexibility in boundary selection. For many applications, the non-proportional-hazards design function `gs_design_nph()` will be used; it calls this function. Initial bound types supported are 1) spending bounds,
2. fixed bounds, and 3) Haybittle-Peto-like bounds. The requirement is to have a boundary update method that can each bound without knowledge of future bounds. As an example, bounds based on conditional power that require knowledge of all future bounds are not supported by this routine; a more limited conditional power method will be demonstrated. Boundary family designs Wang-Tsiatis designs including the original (non-spending-function-based) O'Brien-Fleming and Pocock designs are not supported by `gs_power_npe()`.

## Usage

```

gs_design_npe(
  theta = 0.1,
  theta0 = NULL,
  theta1 = NULL,
  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
)

```

**Arguments**

theta	Natural parameter for group sequential design representing expected incremental drift at all analyses; used for power calculation.
theta0	Natural parameter used for upper bound spending; if NULL, this will be set to 0.
theta1	Natural parameter used for lower bound spending; if NULL, this will be set to theta which yields the usual beta-spending. If set to 0, spending is 2-sided under null hypothesis.
info	Proportionate statistical information at all analyses for input theta.
info0	Proportionate statistical information under null hypothesis, if different than alternative; impacts null hypothesis bound calculation.
info1	Proportionate statistical information under alternate hypothesis; impacts null hypothesis bound calculation.
info_scale	Information scale for calculation. Options are: <ul style="list-style-type: none"> <li>• "h0_h1_info" (default): variance under both null and alternative hypotheses is used.</li> <li>• "h0_info": variance under null hypothesis is used.</li> <li>• "h1_info": variance under alternative hypothesis is used.</li> </ul>
alpha	One-sided Type I error.
beta	Type II error.
upper	Function to compute upper bound.
upar	Parameters passed to the function provided in upper.
lower	Function to compare lower bound.
lpar	Parameters passed to the function provided in 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 FALSE indicates 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).

**Details**

The inputs `info` and `info0` should be vectors of the same length with increasing positive numbers. The design returned will change these by some constant scale factor to ensure the design has power  $1 - \beta$ . The bound specifications in `upper`, `lower`, `upar`, `lpar` will be used to ensure Type I error and other boundary properties are as specified.

**Value**

A tibble with columns analysis, bound, z, probability, theta, info, info0.

**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.

**Author(s)**

Keaven Anderson <keaven\_anderson@merck.com>

**Examples**

```
library(dplyr)
library(gsDesign)

# -----#
#           example 1           #
# -----#
# 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           #
# -----#
# 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 %>% filter(bound == "upper"))$info,
  upper = gs_b,
  upar = (x %>% filter(bound == "upper"))$z,
  lower = gs_b,
  lpar = rep(-Inf, 3)
)

# -----#
#           example 3           #
# -----#
# Spending bound examples
# 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           #
# -----#
# 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           #
# -----#
# 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 %>% filter(bound == "upper"))$z,
  lpar = -(xx %>% filter(bound == "upper"))$z
)

```

---

gs_design_rd	<i>Group sequential design of binary outcome measuring in risk difference</i>
--------------	---

---

## 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"),
  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: <ul style="list-style-type: none"> <li>• "h0_h1_info" (default): variance under both null and alternative hypotheses is used.</li> <li>• "h0_info": variance under null hypothesis is used.</li> <li>• "h1_info": variance under alternative hypothesis is used.</li> </ul>
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
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))
  )

# ----- #
#   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 = wlr_weight_fh,
  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_b,
  upar = gsDesign(k = 3, test.type = 1, n.I = c(0.25, 0.75, 1), sfu = sfLDOF, sfupar =
    NULL)$upper$bound,
  lower = gs_b,
  lpar = c(qnorm(0.1), -Inf, -Inf),
  test_upper = TRUE,
  test_lower = TRUE,
  h1_spending = TRUE,
  r = 18,
  tol = 1e-06,
  interval = c(0.01, 100)
)
```

**Arguments**

enroll_rate	Enrollment rates.
fail_rate	Failure and dropout rates.
weight	Weight of weighted log rank test: <ul style="list-style-type: none"> <li>• "1" = unweighted.</li> <li>• "n" = Gehan-Breslow.</li> <li>• "sqrtN" = Tarone-Ware.</li> <li>• "FH_p[a]_q[b]" = Fleming-Harrington with p=a and q=b.</li> </ul>
approx	Approximate estimation method for Z statistics. <ul style="list-style-type: none"> <li>• "event_driven" = only work under proportional hazard model with log rank test.</li> <li>• "asymptotic".</li> </ul>
alpha	One-sided Type I error.
beta	Type II error.
ratio	Experimental:Control randomization ratio (not yet implemented).
info_frac	Targeted information fraction at each analysis.
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.

analysis_time	Minimum time of analysis.
binding	Indicator of whether futility bound is binding; default of FALSE is recommended.
upper	Function to compute upper bound.
upar	Parameters passed to upper.
lower	Function to compute lower bound.
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 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.
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).
interval	An interval that is presumed to include the time at which expected event count is equal to targeted event.

**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(dplyr)
library(mvtnorm)
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
)

# -----#
#       example 1       #
# -----#
# Boundary is fixed
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
)

gs_design_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  ratio = 1,
  alpha = 0.025, beta = 0.2,
  weight = function(x, arm0, arm1) {
    wlr_weight_fh(x, arm0, arm1, rho = 0, gamma = 0.5)
  },
  upper = gs_b,
  upar = x$upper$bound,
  lower = gs_b,
  lpar = x$lower$bound,
  analysis_time = c(12, 24, 36)
)

# -----#

```

```

#           example 2           #
# ----- #
# Boundary derived by spending function
gs_design_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  ratio = 1,
  alpha = 0.025, beta = 0.2,
  weight = function(x, arm0, arm1) {
    wlr_weight_fh(x, arm0, arm1, rho = 0, gamma = 0.5)
  },
  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(12, 24, 36)
)

```

---

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, 100)
)

```

### Arguments

enroll_rate	Enrollment rates.
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.
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`, `AHR`, `Events`, `theta`, `info`, `info0`. `info`, and `info0` contain statistical information under  $H_1$ ,  $H_0$ , 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 tibble 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))
```

**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 <a href="#">define_enroll_rate()</a> .
fail_rate	A fail_rate data frame with or without stratum created by <a href="#">define_fail_rate()</a> .
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")
)
```

**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 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,
  weight = "ss"
)

# ----- #
#      example 6      #
# ----- #
# 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 = 0.02,
  ratio = 1,
  weight = "invar"
)

# ----- #
#      example 7      #
# ----- #
# stratified case under inverse variance weighting and H0: rd0 != 0 and
# rd0 difference for different statum
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 = wlr_weight_fh,
  approx = "asymptotic",
  interval = c(0.01, 100)
)

```

## Arguments

enroll_rate	An enroll_rate data frame with or without stratum created by <a href="#">define_enroll_rate()</a> .
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: <ul style="list-style-type: none"> <li>• "1" = unweighted.</li> <li>• "n" = Gehan-Breslow.</li> <li>• "sqrtN" = Tarone-Ware.</li> <li>• "FH_p[a]_q[b]" = Fleming-Harrington with p=a and q=b.</li> </ul>
approx	Approximate estimation method for Z statistics. <ul style="list-style-type: none"> <li>• "event_driven" = only work under proportional hazard model with log rank test.</li> <li>• "asymptotic".</li> </ul>
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	<i>Group sequential design power using average hazard ratio under non-proportional hazards</i>
--------------	--

---

### Description

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_b,
  upar = gsDesign(k = length(event), test.type = 1, n.I = event, maxn.IPlan = max(event),
    sfu = sfLDOF, sfupar = NULL)$upper$bound,
  lower = gs_b,
  lpar = c(qnorm(0.1), rep(-Inf, 2)),
  test_lower = TRUE,
  test_upper = TRUE,
  ratio = 1,
  binding = FALSE,
  info_scale = c("h0_h1_info", "h0_info", "h1_info"),
  r = 18,
  tol = 1e-06,
  interval = c(0.01, 100)
)
```

### Arguments

enroll_rate	An enroll_rate data frame with or without stratum created by <a href="#">define_enroll_rate()</a> .
fail_rate	Failure and dropout rates.
event	Targeted event at each analysis.
analysis_time	Minimum time of analysis.
upper	Function to compute upper bound.
upar	Parameters passed to upper.
lower	Function to compute lower bound.
lpar	Parameters passed to lower.
test_lower	Indicator of which analyses should include an 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 (not yet implemented).
binding	Indicator of whether futility bound is binding; default of FALSE is recommended.
info_scale	Information scale for calculation. Options are: <ul style="list-style-type: none"> <li>• "h0_h1_info" (default): variance under both null and alternative hypotheses is used.</li> <li>• "h0_info": variance under null hypothesis is used.</li> <li>• "h1_info": variance under alternative hypothesis is used.</li> </ul>
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).
interval	An interval that is presumed to include the time at which expected event count is equal to targeted event.

### Details

Bound 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.

### Value

A tibble with columns Analysis, Bound, Z, Probability, theta, Time, AHR, Events. Contains a row for each analysis and each bound.

### 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)
library(dplyr)

# 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()

# 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, param = NULL, timing = NULL),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL)
)

# Example 3 -----
# 2-sided symmetric O'Brien-Fleming spending bound, driven by event,
# i.e., `event = c(20, 50, 70)`, `analysis_time = NULL`

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, param = NULL, timing = NULL),
  lower = gs_spending_bound,
  lpar = list(sf = gsDesign::sfLDOF, total_spend = 0.025, param = NULL, timing = NULL)
)

# 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),
  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\_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 = GenzBretz(maxpts = 1e+05, abseps = 1e-05),
  ...
)
```

**Arguments**

enroll_rate	Enrollment rates.
fail_rate	Failure and dropout rates.
fh_test	A data frame to summarize the test in each analysis. See examples for its data structure.
ratio	Experimental:Control randomization ratio (not yet implemented).
binding	Indicator of whether futility bound is binding; default of FALSE is recommended.
upper	Function to compute upper bound.
upar	Parameters passed to upper.
lower	Function to compute lower bound.
lpar	Parameters passed to lower.
algorithm	an object of class <a href="#">GenzBretz</a> , <a href="#">Miwa</a> or <a href="#">TVPACK</a> specifying both the algorithm to be used as well as the associated hyper parameters.
...	Additional parameters passed to <a href="#">mvtnorm::pmvnorm</a> .

**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(dplyr)
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)
)

```

## Description

Derives group sequential bounds and boundary crossing probabilities for a design. It allows a non-constant treatment effect over time, but also can be applied for the usual homogeneous effect size designs. It requires treatment effect and statistical information at each analysis as well as a method of deriving bounds, such as spending. The routine enables two things not available in the gsDesign package:

1. non-constant effect, 2) more flexibility in boundary selection. For many applications, the non-proportional-hazards design function `gs_design_nph()` will be used; it calls this function. Initial bound types supported are 1) spending bounds,
2. fixed bounds, and 3) Haybittle-Peto-like bounds. The requirement is to have a boundary update method that can each bound without knowledge of future bounds. As an example, bounds based on conditional power that require knowledge of all future bounds are not supported by this routine; a more limited conditional power method will be demonstrated. Boundary family designs Wang-Tsiatis designs including the original (non-spending-function-based) O'Brien-Fleming and Pocock designs are not supported by `gs_power_npe()`.

## Usage

```
gs_power_npe(
  theta = 0.1,
  theta0 = NULL,
  theta1 = NULL,
  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

<code>theta</code>	Natural parameter for group sequential design representing expected incremental drift at all analyses; used for power calculation.
<code>theta0</code>	Natural parameter for null hypothesis, if needed for upper bound computation.
<code>theta1</code>	Natural parameter for alternate hypothesis, if needed for lower bound computation.
<code>info</code>	Statistical information at all analyses for input <code>theta</code> .
<code>info0</code>	Statistical information under null hypothesis, if different than <code>info</code> ; impacts null hypothesis bound calculation.

info1	Statistical information under hypothesis used for futility bound calculation if different from info; impacts futility hypothesis bound calculation.
info_scale	Information scale for calculation. Options are: <ul style="list-style-type: none"> <li>• "h0_h1_info" (default): variance under both null and alternative hypotheses is used.</li> <li>• "h0_info": variance under null hypothesis is used.</li> <li>• "h1_info": variance under alternative hypothesis is used.</li> </ul>
upper	Function to compute upper bound.
upar	Parameters passed to upper.
lower	Function to compare lower bound.
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 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).

### Value

A tibble with columns as analysis index, bounds, z, crossing probability, theta (standardized treatment effect), theta1 (standardized treatment effect under alternative hypothesis), information fraction, and statistical information.

### Specification

- 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.

## Examples

```

library(gsDesign)
library(gsDesign2)
library(dplyr)

# Default (single analysis; Type I error controlled)
gs_power_npe(theta = 0) %>% filter(bound == "upper")

# 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)
) %>%
  filter(bound == "upper")

# Fixed bound with 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)
)

# 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)
)

# 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 %>% filter(bound == "upper"))$z,
  lpar = -(x %>% filter(bound == "upper"))$z
)

```

---

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"),
  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: <ul style="list-style-type: none"> <li>• "h0_h1_info" (default): variance under both null and alternative hypotheses is used.</li> <li>• "h0_info": variance under null hypothesis is used.</li> <li>• "h1_info": variance under alternative hypothesis is used.</li> </ul>
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,
upper = gs_b,
lower = gs_b,
upar = gsDesign(k = 3, test.type = 1, n.I = c(30, 40, 50), maxn.IPlan = 50, sfu =
  sflDOF, sfupar = NULL)$upper$bound,
lpar = c(qnorm(0.1), rep(-Inf, 2)),
test_upper = TRUE,
test_lower = TRUE,
ratio = 1,
weight = wlr_weight_fh,
info_scale = c("h0_h1_info", "h0_info", "h1_info"),
approx = "asymptotic",
r = 18,
tol = 1e-06,
interval = c(0.01, 100)
)

```

### Arguments

enroll_rate	Enrollment rates.
fail_rate	Failure and dropout rates.
event	Targeted event at each analysis.
analysis_time	Minimum time of analysis.
binding	Indicator of whether futility bound is binding; default of FALSE is recommended.
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 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 (not yet implemented).
weight	Weight of weighted log rank test: <ul style="list-style-type: none"> <li>• "1" = unweighted.</li> <li>• "n" = Gehan-Breslow.</li> <li>• "sqrtN" = Tarone-Ware.</li> </ul>

	<ul style="list-style-type: none"> <li>• "FH_p[a]_q[b]" = Fleming-Harrington with p=a and q=b.</li> </ul>
info_scale	<p>Information scale for calculation. Options are:</p> <ul style="list-style-type: none"> <li>• "h0_h1_info" (default): variance under both null and alternative hypotheses is used.</li> <li>• "h0_info": variance under null hypothesis is used.</li> <li>• "h1_info": variance under alternative hypothesis is used.</li> </ul>
approx	<p>Approximate estimation method for Z statistics.</p> <ul style="list-style-type: none"> <li>• "event_driven" = only work under proportional hazard model with log rank test.</li> <li>• "asymptotic".</li> </ul>
r	<p>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.</p>
tol	<p>Tolerance parameter for boundary convergence (on Z-scale).</p>
interval	<p>An interval that is presumed to include the time at which expected event count is equal to targeted event.</p>

### 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: <ul style="list-style-type: none"> <li>• <code>sf</code> (class spending function).</li> <li>• <code>total_spend</code> (total spend).</li> <li>• <code>param</code> (any parameters needed by the spending function <code>sf()</code>).</li> </ul> |

	<ul style="list-style-type: none"> <li>• <code>timing</code> (a vector containing values at which spending function is to be evaluated or NULL if information-based spending is used).</li> <li>• <code>max_info</code> (when <code>timing</code> is NULL, this can be input as positive number to be used with <code>info</code> for information fraction at each analysis).</li> </ul>
<code>hgm1</code>	Subdensity grid from <code>h1()</code> ( $k=2$ ) or <code>hupdate()</code> ( $k>2$ ) for analysis $k-1$ ; if $k=1$ , this is not used and may be NULL.
<code>theta</code>	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 ( $\theta = 0$ ).
<code>info</code>	Statistical information at all analyses, at least up to analysis $k$ .
<code>efficacy</code>	TRUE (default) for efficacy bound, FALSE otherwise.
<code>test_bound</code>	A logical vector of the same length as <code>info</code> should indicate which analyses will have a bound.
<code>r</code>	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 <code>r</code> will not be changed by the user.
<code>tol</code>	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)
)
```

---

<code>gs_spending_combo</code>	<i>Derive spending bound for MaxCombo group sequential boundary</i>
--------------------------------	---

---

**Description**

Derive spending bound for MaxCombo group sequential boundary

**Usage**

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

**Arguments**

<code>par</code>	A list with the following items: <ul style="list-style-type: none"> <li>• <code>sf</code> (class spending function).</li> <li>• <code>total_spend</code> (total spend).</li> <li>• <code>param</code> (any parameters needed by the spending function <code>sf()</code>).</li> <li>• <code>timing</code> (a vector containing values at which spending function is to be evaluated or NULL if information-based spending is used).</li> <li>• <code>max_info</code> (when <code>timing</code> is NULL, this can be input as positive number to be used with <code>info</code> for information fraction at each analysis).</li> </ul>
<code>info</code>	Statistical information at all analyses, at least up to analysis <code>k</code> .
<code>...</code>	Additional parameters passed to <code>par\$sf</code> .

**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)

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

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  $\lambda_i$  is the failure rate in the interval  $(t_{i-1}, t_i]$ ,  $i = 1, 2, \dots, M$  where  $0 = t_0 < t_1 < \dots < t_M = \infty$ . 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 (test version)*

---

## 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 <code>define_enroll_rate()</code> .
fail_rate	A fail_rate data frame with or without stratum created by <code>define_fail_rate()</code> .
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	Ratio of experimental to control randomization.

**Value**

A tibble with time (from total\_duration), ahr (average hazard ratio), 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 tibble 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: 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 = 1,
  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))
```

---

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"),
  ...
)
```

**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.

col\_vars            The variables to be displayed.  
col\_decimals        The decimals to be displayed for the displayed variables in col\_vars.  
bound\_names         Names for bounds; default is c("Efficacy", "Futility").

**Value**

A summary table (data frame).

**Examples**

```
library(dplyr)

# 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)
library(dplyr)

# 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)

```

```

# weight function in WLR
wgt00 <- function(x, arm0, arm1) {
  wlr_weight_fh(x, arm0, arm1, rho = 0, gamma = 0)
}
wgt05 <- function(x, arm0, arm1) {
  wlr_weight_fh(x, arm0, arm1, rho = 0, gamma = .5)
}

# 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)
)

# ----- #
#           ahr           #
# ----- #

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()
x_ahr %>% summary(analysis_vars = c("time", "event", "info_frac"), analysis_decimals = c(1, 0, 2))
x_ahr %>% summary(bound_names = c("A is better", "B is better"))

# ----- #
#           wlr           #
# ----- #

x_wlr <- gs_design_wlr(
  enroll_rate = enroll_rate,
  fail_rate = fail_rate,
  weight = wgt05,
  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()

```

**Description**

Rounds sample size to an even number for equal design

**Usage**

```
to_integer(x, ...)  
  
## S3 method for class 'fixed_design'  
to_integer(x, sample_size = TRUE, ...)  
  
## S3 method for class 'gs_design'  
to_integer(x, sample_size = TRUE, ...)
```

**Arguments**

x	An object returned by fixed_design_xxx() and gs_design_xxx().
...	Additional parameters (not used).
sample_size	Logical, indicating if ceiling sample size to an even integer.

**Value**

A list similar to the output of fixed\_design\_xxx() and gs\_design\_xxx(), except the sample size is an integer.

**Examples**

```
library(dplyr)  
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()  
  
# 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()

# 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 = 4,
  study_duration = 36, ratio = 1
)
x %>% to_integer()

gs_design_ahr() %>% to_integer()

gs_design_wlr() %>% to_integer()

```

---

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, wmax = 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.
wmax	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 `npsurvSS::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, wmax = 1.2)

```

# Index

ahr, 3  
ahr(), 14, 48, 54, 56  
as\_gt, 5

define\_enroll\_rate, 9  
define\_enroll\_rate(), 3, 11, 12, 14, 49, 53, 55, 78  
define\_fail\_rate, 9  
define\_fail\_rate(), 3, 12, 14, 49, 78

expected\_accrual, 10  
expected\_event, 12  
expected\_time, 14  
expected\_time(), 48, 54, 56

fixed\_design\_ahr, 15  
fixed\_design\_fh, 17  
fixed\_design\_lf, 18  
fixed\_design\_maxcombo, 20  
fixed\_design\_mb, 21  
fixed\_design\_milestone, 23  
fixed\_design\_rd, 24  
fixed\_design\_rmst, 25

GenzBretz, 34, 58  
gs\_b, 27  
gs\_create\_arm, 28  
gs\_design\_ahr, 29  
gs\_design\_combo, 33  
gs\_design\_npe, 36  
gs\_design\_rd, 41  
gs\_design\_wlr, 43  
gs\_info\_ahr, 47  
gs\_info\_combo, 49  
gs\_info\_rd, 50  
gs\_info\_wlr, 53  
gs\_power\_ahr, 55  
gs\_power\_combo, 57  
gs\_power\_npe, 59  
gs\_power\_npe(), 36  
gs\_power\_rd, 63  
gs\_power\_wlr, 68  
gs\_spending\_bound, 73  
gs\_spending\_combo, 75

Miwa, 34, 58  
mvtnorm::pmvnorm, 34, 58

ppwe, 76  
pw\_info, 77

s2pwe, 79  
summary.fixed\_design, 80  
summary.gs\_design  
    (summary.fixed\_design), 80

to\_integer, 84  
TVPACK, 34, 58

wlr\_weight, 86  
wlr\_weight\_1(wlr\_weight), 86  
wlr\_weight\_fh(wlr\_weight), 86  
wlr\_weight\_mb(wlr\_weight), 86  
wlr\_weight\_n(wlr\_weight), 86