

Package ‘SimNPH’

September 19, 2025

Type Package

Title Simulate Non-Proportional Hazards

Version 0.5.8

License BSL-1.0

Description A toolkit for simulation studies concerning time-to-event endpoints with non-proportional hazards. 'SimNPH' encompasses functions for simulating time-to-event data in various scenarios, simulating different trial designs like fixed-followup, event-driven, and group sequential designs. The package provides functions to calculate the true values of common summary statistics for the implemented scenarios and offers common analysis methods for time-to-event data. Helper functions for running simulations with the 'SimDesign' package and for aggregating and presenting the results are also included. Results of the conducted simulation study are available in the paper: ``A Comparison of Statistical Methods for Time-To-Event Analyses in Randomized Controlled Trials Under Non-Proportional Hazards'', Klingmüller et al. (2025) <[doi:10.1002/sim.70019](https://doi.org/10.1002/sim.70019)>.

Encoding UTF-8

LazyData true

Depends SimDesign, survival, R (>= 4.1.0)

Suggests knitr, rmarkdown, testthat (>= 3.0.0), covr, withr, ldbounds, ggplot2, patchwork, vdiffR

Imports miniPCH (>= 0.3.0), purrr (>= 1.0.0), rlang, tibble, nph, nphRCT, car, dplyr, stringr, methods, tidyr

Config/testthat/edition 3

VignetteBuilder knitr

RoxygenNote 7.3.3

URL <https://simnph.github.io/SimNPH/>,
<https://github.com/SimNPH/SimNPH/>

BugReports <https://github.com/SimNPH/SimNPH/issues/>

NeedsCompilation no

Author Tobias Fellingner [aut, cre] (ORCID:
<https://orcid.org/0000-0001-9474-2731>),
 Florian Klinglmueller [aut] (ORCID:
<https://orcid.org/0000-0002-7346-3669>)

Maintainer Tobias Fellingner <tobias.fellinger@ages.at>

Repository CRAN

Date/Publication 2025-09-19 10:00:02 UTC

Contents

analyse_aft	3
analyse_ahr	4
analyse_coxph	5
analyse_describe	6
analyse_diff_median_survival	7
analyse_gehan_wilcoxon	9
analyse_group_sequential	9
analyse_logrank	11
analyse_logrank_fh_weights	12
analyse_maxcombo	13
analyse_milestone_survival	13
analyse_modelstly_weighted	15
analyse_piecewise_exponential	16
analyse_rmst_diff	17
analyse_weibull	18
assumptions_progression	19
combination_tests_delayed	22
create_summarise_function	23
design_fixed_followup	24
design_group_sequential	25
generate_crossing_hazards	25
generate_delayed_effect	28
generate_subgroup	31
labs_from_labels	34
mixture_haz_fun	35
progression_cdf_fun	38
r2m	40
random_censoring_exp	41
recruitment_uniform	42
rename_results_column	44
results_pivot_longer	46
shhr_gg	49
summarise_estimator	50
summarise_test	52
upsert_merge	53
wrap_all_in_trycatch	55

analyse_aft	<i>Analyse Dataset with accelerated failure time models</i>
-------------	---

Description

Analyse Dataset with accelerated failure time models

Usage

```
analyse_aft(level = 0.95, dist = "weibull", alternative = "two.sided")
```

Arguments

level	confidence level for CI computation
dist	passed to survival::survreg
alternative	alternative hypothesis for the tests "two.sided" or "one.sided"

Details

alternative can be "two.sided" for a two sided test of equality of the summary statistic or "one.sided" for a one sided test testing H0: treatment has equal or shorter survival than control vs. H1 treatment has longer survival than control.

Value

an analyse function that returns a list with the elements

- p p value of the score test (two.sided) or the Wald test (one.sided)
- alternative the alternative used
- coef coefficient for trt
- lower lower 95% confidence intervall boundary for the coefficient
- upperlower 95% confidence intervall boundary for the coefficient
- CI_level the CI level used
- N_pat number of patients
- N_evt number of events

Examples

```
condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by = NULL
) |>
  head(1)
dat <- generate_delayed_effect(condition)
analyse_aft()(condition, dat)
analyse_aft(dist="lognormal")(condition, dat)
```

analyse_ahr

Analyse the dataset using estimators for the the average hazard ratio

Description

Analyse the dataset using estimators for the the average hazard ratio

Usage

```
analyse_ahr(
  max_time = NA,
  type = "AHR",
  level = 0.95,
  alternative = "two.sided"
)
```

Arguments

max_time	time for which the AHR is calculated
type	"AHR" for average hazard ratio "gAHR" for geometric average hazard ratio
level	confidence level for CI computation
alternative	alternative hypothesis for the tests "two.sided" or "one.sided"

Details

The implementation from the `nph` package is used, see the documentation there for details.

alternative can be "two.sided" for a two sided test of equality of the summary statistic or "one.sided" for a one sided test testing H0: treatment has equal or shorter survival than control vs. H1 treatment has longer survival than control.

The data.frame returned by the created function includes the following columns:

- p p value of the test, see Details
- alternative the alternative used
- AHR/gAHR estimated (geometric) average hazard ratio
- AHR_lower/gAHR_lower unadjusted lower bound of the confidence interval for the (geometric) average hazard ratio
- AHR_upper/gAHR_upper unadjusted upper bound of the confidence interval for the (geometric) average hazard ratio
- CI_level the CI level used
- N_pat number of patients
- N_evt number of events

Value

Returns an analysis function, that can be used in `runSimulations`

See Also[nph::nphparams](#)**Examples**

```

condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by = NULL
) |>
  head(1)
dat <- generate_delayed_effect(condition)
analyse_ahr()(condition, dat)
analyse_ahr(type = "gAHR")(condition, dat)
analyse_ahr(max_time = 50, type = "AHR")(condition, dat)
analyse_ahr(max_time = 50, type = "gAHR")(condition, dat)

```

analyse_coxph

*Analyse Dataset with the Cox Proportional Hazards Model***Description**

Analyse Dataset with the Cox Proportional Hazards Model

Usage

```
analyse_coxph(level = 0.95, alternative = "two.sided")
```

Arguments

level	confidence level for CI computation
alternative	alternative hypothesis for the tests "two.sided" or "one.sided"

Details

alternative can be "two.sided" for a two sided test of equality of the summary statistic or "one.sided" for a one sided test testing H0: treatment has equal or shorter survival than control vs. H1 treatment has longer survival than control.

Value

an analyse function that returns a list with the elements

- p p value of the score test (two.sided) or the Wald test (one.sided)
- alternative the alternative used
- coef coefficient for trt
- hr hazard ratio for trt

- hr_lower lower 95% confidence interval boundary for the hazard ratio for trt
- hr_upper upper 95% confidence interval boundary for the hazard ratio for trt
- CI_level the CI level used
- N_pat number of patients
- N_evt number of events

Examples

```
condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by = NULL
) |>
  head(1)
dat <- generate_delayed_effect(condition)
analyse_coxph()(condition, dat)
```

analyse_describe

Create a Function for Descriptive Statistics of a Dataset

Description

Create a Function for Descriptive Statistics of a Dataset

Usage

```
analyse_describe()

summarise_describe(name = NULL)
```

Arguments

name	name for the summarise function, appended to the name of the analysis method in the final results
------	---

Value

an analyse function that returns a list with the elements

- followup follow up time
- events table of events vs. treatment
- ice if column ice is present, table of intercurrent events, events, treatment
- subgroup if column subgroup is present, table of subgroup, events, treatment

A function that can be used in Summarise that returns a data frame with columns with means and standard deviations for every variable in the description.

Functions

- summarise_describe(): Summarise Descriptive Statistics

Examples

```

condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
) |>
head(1)
dat <- generate_delayed_effect(condition)
analyse_describe()(condition, dat)
condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
) |>
tail(4) |>
head(1)

summarise_all <- create_summarise_function(
  describe=summarise_describe()
)

# runs simulations
sim_results <- runSimulation(
  design=condition,
  replications=100,
  generate=generate_delayed_effect,
  analyse=list(
    describe=analyse_describe()
  ),
  summarise = summarise_all
)

# study time is missing, since there was no admin. censoring
sim_results[, 9:16]
```

analyse_diff_median_survival

Analyse the dataset using difference in median survival

Description

Analyse the dataset using difference in median survival

Usage

```
analyse_diff_median_survival(
  quant = 0.5,
  level = 0.95,
  alternative = "two.sided"
)
```

Arguments

quant	quantile for which the difference should be calculated, defaults to the median
level	confidence level for CI computation
alternative	alternative hypothesis for the tests "two.sided" or "one.sided"

Details

The implementation from the `nph` package is used, see the documentation there for details.

The `data.frame` returned by the created function includes the following columns:

- `p` p value of the test, see Details
- `alternative` the alternative used
- `diff_Q` estimated difference in quantile of the survival functions
- `diff_Q_lower` unadjusted lower bound of the confidence interval for the difference in quantile of the survival functions
- `diff_Q_upper` unadjusted upper bound of the confidence interval for the difference in quantile of the survival functions
- `CI_level` the CI level used
- `quantile` quantile used for estimation
- `N_pat` number of patients
- `N_evt` number of events

Value

Returns an analysis function, that can be used in `runSimulations`

See Also

[nph::nphparams](#)

Examples

```
condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by = NULL
) |>
  head(1)
dat <- generate_delayed_effect(condition)
analyse_diff_median_survival()(condition, dat)
```

`analyse_gehan_wilcoxon`*Create Analyse function for Gehan Wilcoxon test*

Description

Create Analyse function for Gehan Wilcoxon test

Usage

```
analyse_gehan_wilcoxon(alternative = "two.sided")
```

Arguments

`alternative` alternative hypothesis for the tests "two.sided" or "one.sided"

Details

`alternative` can be "two.sided" for a two sided test of equality of the summary statistic or "one.sided" for a one sided test testing H0: treatment has equal or shorter survival than control vs. H1 treatment has longer survival than control.

Value

an analyse function that can be used in `runSimulation`

Examples

```
condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by = NULL
) |>
  head(1)
dat <- generate_delayed_effect(condition)
analyse_gehan_wilcoxon()(condition, dat)
```

`analyse_group_sequential`*Create Analyse Functions for Group Sequential Design*

Description

Create Analyse Functions for Group Sequential Design

Summarise Output from Analyse Functions for Group Sequential Design

Usage

```
analyse_group_sequential(followup, followup_type, alpha, analyse_functions)

summarise_group_sequential(name = NULL)
```

Arguments

followup	followup events or time
followup_type	"events" or "time"
alpha	nominal alpha at each stage
analyse_functions	analyse function or list of analyse functions
name	name attribute of the returned closure

Details

followup, followup_type and alpha are evaluated for every simulated dataset, i.e. the arguments to the Analyse function are available, expressions like followup=c(condition\$interim, condition\$max_followup) are valid arguments.

analyse_functions should take arguments condition, dataset and fixed_objects and return a list containing p-value, number of patients and number of event in the columns p, N_pat and N_evt.

Value

an analyse function that can be used in runSimulation

Returns a function with the arguments:

- condition
- results
- fixed objects

that can be passed to create_summarise_function or to SimDesign::runSimulation and that returns a data.frame.

Functions

- summarise_group_sequential(): Summarise Output from Analyse Functions for Group Sequential Design

Examples

```
# create a function to analyse after interim_events and maximum followup time
# given in the condition row of the design data.frame with given
# nominal alpha
analyse_maxcombo_sequential <- analyse_group_sequential(
  followup = c(condition$interim_events, condition$followup),
  followup_type = c("event", "time"),
  alpha = c(0.025, 0.05),
```

```

    analyse_functions = analyse_maxcombo()
  )
  Summarise <- create_summarise_function(
    maxcombo_seq = summarise_group_sequential(),
    logrank_seq = summarise_group_sequential(name="logrank")
  )

```

analyse_logrank

Analyse Dataset with the Logrank Test

Description

Analyse Dataset with the Logrank Test

Usage

```
analyse_logrank(alternative = "two.sided")
```

Arguments

alternative alternative hypothesis for the tests "two.sided" or "one.sided"

Details

alternative can be "two.sided" for a two sided test of equality of the summary statistic or "one.sided" for a one sided test testing H0: treatment has equal or shorter survival than control vs. H1 treatment has longer survival than control.

Value

an analysis function that returns a data.frame with the columns

- p-value of the logrank test
- alternative the alternative used
- N_pat number of patients
- N_evt number of events

Examples

```

condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by = NULL
) |>
  head(1)
dat <- generate_delayed_effect(condition)
analyse_logrank()(condition, dat)

```

 analyse_logrank_fh_weights

Analyse Dataset with the Fleming Harrington weighted Logrank Test

Description

Analyse Dataset with the Fleming Harrington weighted Logrank Test

Usage

```
analyse_logrank_fh_weights(rho, gamma, alternative = "two.sided")
```

Arguments

rho	rho for the rho-gamma family of weights
gamma	gamma for the rho-gamma family of weights
alternative	alternative hypothesis for the tests "two.sided" or "one.sided"

Details

alternative can be "two.sided" for a two sided test of equality of the summary statistic or "one.sided" for a one sided test testing H0: treatment has equal or shorter survival than control vs. H1 treatment has longer survival than control.

Value

a function with the arguments condition, dat and fixed_objects that returns a dataframe with the p-value of the weighted logrank test in the column p. See ?SimDesign::Analyse for details on the arguments condition, dat, fixed_arguments.

Examples

```
condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by = NULL
) |>
  head(1)
dat <- generate_delayed_effect(condition)
# create two functions with different weights
analyse_01 <- analyse_logrank_fh_weights(rho = 0, gamma = 1)
analyse_10 <- analyse_logrank_fh_weights(rho = 1, gamma = 0)
# run the tests created before
analyse_01(condition, dat)
analyse_10(condition, dat)
```

analyse_maxcombo

Analyse Dataset with the Maxcombo Test

Description

Analyse Dataset with the Maxcombo Test

Usage

```
analyse_maxcombo(alternative = "two.sided")
```

Arguments

alternative alternative hypothesis for the tests "two.sided" or "one.sided"

Details

alternative can be "two.sided" for a two sided test of equality of the summary statistic or "one.sided" for a one sided test testing H0: treatment has equal or shorter survival than control vs. H1 treatment has longer survival than control.

Value

an analyse function that returns a data.frame with the combined p-value of the max combo test in the column p

Examples

```
condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by = NULL
) |>
  head(1)
dat <- generate_delayed_effect(condition)
analyse_maxcombo()(condition, dat)
```

analyse_milestone_survival

Analyse the Dataset using difference or quotient of milestone survival

Description

Analyse the Dataset using difference or quotient of milestone survival

Usage

```
analyse_milestone_survival(
  times,
  what = "quot",
  level = 0.95,
  alternative = "two.sided"
)
```

Arguments

<code>times</code>	followup times at which the the survival should be compared
<code>what</code>	"quot" for quotient and "diff" for difference of survival probabilities
<code>level</code>	confidence level for CI computation
<code>alternative</code>	alternative hypothesis for the tests "two.sided" or "one.sided"

Details

The implementation from the `nph` package is used, see the documentation there for details.

`alternative` can be "two.sided" for a two sided test of equality of the summary statistic or "one.sided" for a one sided test testing H_0 : treatment has equal or shorter survival than control vs. H_1 treatment has longer survival than control.

The data.frame returned by the created function includes the following columns:

- `milestone_surv_ratio / milestone_surv_diff` ratio or difference of survival probabilities
- `times` followup times at which the the survival are compared
- `N_pat` number of patients
- `N_evt` number of events
- `p` p value for the H_0 that the ratios are 1 or the difference is 0 respectively
- `alternative` the alternative used
- `milestone_surv_ratio_lower / milestone_surv_diff_lower` upper/lower CI for the estimate
- `milestone_surv_ratio_upper / milestone_surv_diff_upper` upper/lower CI for the estimate
- `CI_level` the CI level used

Value

Returns an analysis function, that can be used in `runSimulations`

See Also

[nph::nphparams](#)

Examples

```

condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
) |>
  head(1)
dat <- generate_delayed_effect(condition)
analyse_milestone_survival(3:5)(condition, dat)
analyse_milestone_survival(3:5, what="diff")(condition, dat)

```

analyse_modelstly_weighted

Create Analyse function for the modestly weighted logrank test

Description

Create Analyse function for the modestly weighted logrank test

Usage

```
analyse_modelstly_weighted(t_star)
```

Arguments

t_star parameter t* of the modestly weighted logrank test

Value

an analyse function that can be used in runSimulation

Examples

```

condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
) |>
  head(1)
dat <- generate_delayed_effect(condition)
analyse_modelstly_weighted(20)(condition, dat)

```

`analyse_piecewise_exponential`*Create Analyse function for piecewise exponential model*

Description

Create Analyse function for piecewise exponential model

Usage

```
analyse_piecewise_exponential(cuts, testing_only = FALSE)
```

Arguments

<code>cuts</code>	interval boundaries for the piecewise exponential model
<code>testing_only</code>	if set to TRUE omits all statistics in the intervals and just returns the p value of the global test.

Details

If there's any time interval no patients ever enter, NA is returned for all time intervals. This behavior will likely change in future package versions.

Value

an analyse function that can be used in `runSimulation`

Examples

```
condition <- merge(  
  assumptions_delayed_effect(),  
  design_fixed_followup(),  
  by=NULL  
) |>  
  head(1)  
dat <- generate_delayed_effect(condition)  
analyse_piecewise_exponential(cuts=c(90, 360))(condition, dat)
```

analyse_rmst_diff	Analyse the Dataset using the difference in RMST
-------------------	--

Description

Analyse the Dataset using the difference in RMST

Usage

```
analyse_rmst_diff(max_time = NA, level = 0.95, alternative = "two.sided")
```

Arguments

max_time	time for which the RMST is calculated
level	confidence level for CI computation
alternative	alternative hypothesis for the tests "two.sided" or "one.sided"

Details

The implementation from the `nph` package is used, see the documentation there for details.

`alternative` can be "two.sided" for a two sided test of equality of the summary statistic or "one.sided" for a one sided test testing H0: treatment has equal or shorter survival than control vs. H1 treatment has longer survival than control.

The data.frame returned by the created function includes the following columns:

- `p` p value of the test, see Details
- `alternative` the alternative used
- `rmst_diff` estimated difference in RMST
- `rmst_diff_lower` unadjusted lower bound of the confidence interval for difference in RMST
- `rmst_diff_upper` unadjusted upper bound of the confidence interval for difference in RMST
- `CI_level` the CI level used
- `N_pat` number of patients
- `N_evt` number of events

Value

Returns an analysis function, that can be used in `runSimulations`

See Also

[nph::nphparams](#)

Examples

```

condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by = NULL
) |>
  head(1)
dat <- generate_delayed_effect(condition)
analyse_rmst_diff()(condition, dat)

```

analyse_weibull

*Analyse Dataset with Weibull Regression***Description**

Analyse Dataset with Weibull Regression

Usage

```
analyse_weibull(level = 0.95, alternative = "two.sided")
```

Arguments

level	confidence level for CI computation
alternative	alternative hypothesis for the tests "two.sided" or "one.sided"

Details

the columns in the return are the two-sided p-value for the test of equal medians. The estimated medians in the treatment and control group and the estimated difference in median survival with confidence intervals.

The estimates and tests are constructed by fitting separate Weibull regression models in the treatment and control groups and then estimating the medians and respective variances with the delta-method.

Value

an analysis function that returns a data.frame

Examples

```

condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
) |>
  head(3) |>
  tail(1)

```

```
dat <- generate_delayed_effect(condition)
analyse_weibull()(condition, dat)
```

```
assumptions_progression
```

Create an empty assumptions data.frame for generate_progression

Description

Create an empty assumptions data.frame for generate_progression
 Generate Dataset with changing hazards after disease progression
 Calculate progression rate from proportion of patients who progress
 Calculate hr after onset of treatment effect

Usage

```
assumptions_progression(print = interactive())

generate_progression(condition, fixed_objects = NULL)

true_summary_statistics_progression(
  Design,
  what = "os",
  cutoff_stats = NULL,
  fixed_objects = NULL,
  milestones = NULL
)

progression_rate_from_progression_prop(design)

cen_rate_from_cen_prop_progression(design)

hazard_before_progression_from_PH_effect_size(
  design,
  target_power_ph = NA_real_,
  final_events = NA_real_,
  target_alpha = 0.025
)
```

Arguments

print	print code to generate parameter set?
condition	condition row of Design dataset
fixed_objects	additional settings, see details
Design	Design data.frame for subgroup

what	True summary statistics for which estimand
cutoff_stats	(optionally named) cutoff time, see details
milestones	(optionally named) vector of times at which milestone survival should be calculated
design	design data.frame
target_power_ph	target power under proportional hazards
final_events	target events for inversion of Schönfeld Formula, defaults to condition\$final_events
target_alpha	target one-sided alpha level for the power calculation

Details

assumptions_progression generates a default design data.frame for use with generate_progression. If print is TRUE code to produce the template is also printed for copying, pasting and editing by the user. (This is the default when run in an interactive session.)

Condition has to contain the following columns:

- n_trt number of patients in treatment arm
- n_ctrl number of patients in control arm
- hazard_ctrl hazard in the control arm
- hazard_trt hazard in the treatment arm for not cured patients
- hazard_after_prog hazard after disease progression
- prog_rate_ctrl hazard rate for disease progression under control
- prog_rate_trt hazard rate for disease progression under treatment

what can be "os" for overall survival and "pfs" for progression free survival.

The if fixed_objects contains t_max then this value is used as the maximum time to calculate function like survival, hazard, ... of the data generating models. If this is not given t_max is chosen as the minimum of the $1-(1/10000)$ quantile of all survival distributions in the model.

cutoff_stats are the times used to calculate the statistics like average hazard ratios and RMST, that are only calculated up to a certain point.

For progression_rate_from_progression_prop, the design data.frame, has to contain the columns prog_prop_trt and prog_prop_ctrl with the proportions of patients, who progress in the respective arms.

cen_rate_from_cen_prop_progression takes the proportion of censored patients from the column censoring_prop. This column describes the proportion of patients who are censored randomly before experiencing an event, without regard to administrative censoring.

hazard_before_progression_from_PH_effect_size calculates the hazard ratio after onset of treatment effect as follows: First calculate the hazard in the control arm that would give the same median survival under an exponential model. Then calculate the median survival in the treatment arm that would give the desired power of the logrank test under exponential models in control and treatment arm. Then calibrate the hazard before progression in the treatment arm to give the same median survival time.

This is a heuristic and to some extent arbitrary approach to calculate hazard ratios that correspond to reasonable and realistic scenarios.

Value

For `generate_progression`: a design tibble with default values invisibly

For `generate_progression`: A dataset with the columns `t` (time) and `trt` (1=treatment, 0=control), `evt` (event, currently TRUE for all observations), `t_ice` (time of intercurrent event), `ice` (intercurrent event)

For `true_summary_statistics_subgroup`: the design data.frame passed as argument with the additional columns

For `progression_rate_from_progression_prop`: the design data.frame passed as argument with the additional columns `prog_rate_trt`, `prog_rate_ctrl`

for `cen_rate_from_cen_prop_progression`: design data.frame with the additional column `random_withdrawal`

For `hazard_before_progression_from_PH_effect_size`: the design data.frame passed as argument with the additional column `hazard_trt`.

Functions

- `assumptions_progression()`: generate default assumptions data.frame
- `generate_progression()`: simulates a dataset with changing hazards after disease progression
- `true_summary_statistics_progression()`: calculate true summary statistics for scenarios with disease progression
- `progression_rate_from_progression_prop()`: Calculate progression rate from proportion of patients who progress
- `cen_rate_from_cen_prop_progression()`: calculate censoring rate from censoring proportion
- `hazard_before_progression_from_PH_effect_size()`: Calculate hazard in the treatment arm before progression from PH effect size

Examples

```
Design <- assumptions_progression()
Design
one_simulation <- merge(
  assumptions_progression(),
  design_fixed_followup(),
  by=NULL
) |>
tail(1) |>
generate_progression()
head(one_simulation)
tail(one_simulation)

my_design <- merge(
  assumptions_progression(),
  design_fixed_followup(),
  by=NULL
)
```

```

my_design_os <- true_summary_statistics_progression(my_design, "os")
my_design_pfs <- true_summary_statistics_progression(my_design, "pfs")
my_design_os
my_design_pfs
my_design <- merge(
  assumptions_progression(),
  design_fixed_followup(),
  by=NULL
)
my_design$prog_rate_ctrl <- NA_real_
my_design$prog_rate_trt <- NA_real_
my_design$prog_prop_trt <- 0.2
my_design$prog_prop_ctrl <- 0.3
my_design <- progression_rate_from_progression_prop(my_design)
my_design
design <- expand.grid(
  hazard_ctrl      = m2r(15),          # hazard under control
  hazard_trt       = m2r(18),          # hazard under treatment
  hazard_after_prog = m2r(3),          # hazard after progression
  prog_rate_ctrl    = m2r(12),          # hazard for disease progression under control
  prog_rate_trt     = m2r(c(12,16,18)), # hazard for disease progression under treatment
  censoring_prop    = 0.1,             # rate of random withdrawal
  followup          = 100,             # follow up time
  n_trt             = 50,              # patients in treatment arm
  n_ctrl            = 50               # patients in control arm
)
cen_rate_from_cen_prop_progression(design)

my_design <- merge(
  design_fixed_followup(),
  assumptions_progression(),
  by=NULL
)

my_design$hazard_trt <- NULL
my_design$final_events <- ceiling(0.75 * (my_design$n_trt + my_design$n_ctrl))

my_design <- hazard_before_progression_from_PH_effect_size(my_design, target_power_ph=0.7)
my_design

```

combination_tests_delayed

Results of an example simulation

Description

Results of an example simulation study comparing the power of logrank max-combo and modelstly weighted logrank test in differnt scenarios with delayed onset of treatment effect.

Usage

combination_tests_delayed

Format

a tibble as returned by `SimDesign::runSimulation`.

`create_summarise_function`

Create a summarise function from a named list of functions

Description

Create a summarise function from a named list of functions

Usage

```
create_summarise_function(...)
```

Arguments

... summarise function

Details

the names of the list of functions correspond to the names in the list of analyse functions, each summarise function is applied to the results of the analyse function of the same name, names not present in both lists are omitted in either list.

The functions in the list should have the arguments `condition`, `results` and `fixed_objects`. `results` is a list of lists. The outer list has one element for each replication, the inner list has one entry for each Analyse function. (Analyse functions have to return lists for this to work, otherwise the results are simplified to data.frames. Analyse functions from the `SimNPH` package all return lists.)

The individual summarise functions have to return data.frames, which are concatenated column-wise to give one row per condition. The names of the analyse methods are prepended to the respective column names, if the functions have a "name" attribute this is appended to the column names of the output. Column names not unique after that are appended numbers by `make.unique`.

Value

a function with arguments `condition`, `results`, `fixed_objects`

Examples

```
Summarise <- create_summarise_function(
  maxcombo = function(condition, results, fixed_objects=NULL){
    data.frame("rejection"=mean(results$p < alpha))
  },
  logrank = function(condition, results, fixed_objects=NULL){
    data.frame("rejection"=mean(results$p < alpha))
  }
)
```

design_fixed_followup *Create a data.frame with an example fixed design*

Description

Create a data.frame with an example fixed design

Usage

```
design_fixed_followup(print = interactive())
```

Arguments

print print code to generate parameter set?

Details

design_fixed_followup generates a default design data.frame for use with generate_delayed_effect or other generate_... functions. If print is TRUE code to produce the template is also printed for copying, pasting and editing by the user. (This is the default when run in an interactive session.)

Value

For design_fixed_followup: a design tibble with default values invisibly

Functions

- design_fixed_followup(): generate default fixed design

Examples

```
Design <- design_fixed_followup()
Design
```

 design_group_sequential

Create a data.frame with an example group sequential design

Description

Create a data.frame with an example group sequential design

Usage

```
design_group_sequential(print = interactive())
```

Arguments

print print code to generate parameter set?

Details

design_group_sequential generates a default design data.frame for use with generate_delayed_effect or other generate_... functions. If print is TRUE code to produce the template is also printed for copy-pasting, pasting and editing by the user. (This is the default when run in an interactive session.)

Value

For design_group_sequential: a design tibble with default values invisibly

Functions

- design_group_sequential(): generate default group sequential design

Examples

```
Design <- design_group_sequential()
Design
```

 generate_crossing_hazards

Generate Dataset with crossing hazards

Description

Generate Dataset with crossing hazards

Create an empty assumptions data.frame for generate_crossing_hazards

Calculate hr after crossing the hazard functions

Calculate true summary statistics for scenarios with crossing hazards

Usage

```

generate_crossing_hazards(condition, fixed_objects = NULL)

assumptions_crossing_hazards(print = interactive())

hr_after_crossing_from_PH_effect_size(
  design,
  target_power_ph = NA_real_,
  final_events = NA_real_,
  target_alpha = 0.025
)

cen_rate_from_cen_prop_crossing_hazards(design)

true_summary_statistics_crossing_hazards(
  Design,
  cutoff_stats = NULL,
  milestones = NULL,
  fixed_objects = NULL
)

```

Arguments

<code>condition</code>	condition row of Design dataset
<code>fixed_objects</code>	additional settings, see details
<code>print</code>	print code to generate parameter set?
<code>design</code>	design data.frame
<code>target_power_ph</code>	target power under proportional hazards
<code>final_events</code>	target events for inversion of Schönfeld Formula, defaults to <code>condition\$final_events</code>
<code>target_alpha</code>	target one-sided alpha level for the power calculation
<code>Design</code>	Design data.frame for crossing hazards
<code>cutoff_stats</code>	(optionally named) cutoff time, see details
<code>milestones</code>	(optionally named) vector of times at which milestone survival should be calculated

Details

Condition has to contain the following columns:

- `n_trt` number of patients in treatment arm
- `n_ctrl` number of patients in control arm
- `crossing` time of crossing of the hazards
- `hazard_ctrl` hazard in the control arm = hazard before onset of treatment effect
- `hazard_trt_before` hazard in the treatment arm before onset of treatment effect

- hazard_trt_after hazard in the treatment arm after onset of treatment effect

If `fixed_objects` is given and contains an element `t_max`, then this is used as the cutoff for the simulation used internally. If `t_max` is not given in this way the 1-(1/10000) quantile of the survival distribution in the control or treatment arm is used (which ever is larger).

`assumptions_crossing_hazards` generates a default design data.frame for use with `generate_crossing_hazards`. If `print` is `TRUE` code to produce the template is also printed for copying, pasting and editing by the user. (This is the default when run in an interactive session.)

`hr_after_crossing_from_PH_effect_size` calculates the hazard ratio after crossing of hazards as follows: First, the hazard ratio needed to achieve the desired power under proportional hazards is calculated by inverting Schönfeld's sample size formula. Second the median survival times for both arm under this hazard ratio and proportional hazards are calculated. Finally the hazard rate of the treatment arm after crossing of hazards is set such that the median survival time is the same as the one calculated under proportional hazards.

This is a heuristic and to some extent arbitrary approach to calculate hazard ratios that correspond to reasonable and realistic scenarios.

`cen_rate_from_cen_prop_crossing_hazards` takes the proportion of censored patients from the column `censoring_prop`. This column describes the proportion of patients who are censored randomly before experiencing an event, without regard to administrative censoring.

`cutoff_stats` are the times used to calculate the statistics like average hazard ratios and RMST, that are only calculated up to a certain point.

Value

For `generate_crossing_hazards`: A dataset with the columns `t` (time) and `trt` (1=treatment, 0=control), `evt` (event, currently `TRUE` for all observations)

For `assumptions_crossing_hazards`: a design tibble with default values invisibly

For `hr_after_crossing_from_PH_effect_size`: the design data.frame passed as argument with the additional column `hazard_trt`.

for `cen_rate_from_cen_prop_crossing_hazards`: design data.frame with the additional column `random_withdrawal`

For `true_summary_statistics_crossing_hazards`: the design data.frame passed as argument with additional columns,

Functions

- `generate_crossing_hazards()`: simulates a dataset with crossing hazards
- `assumptions_crossing_hazards()`: generate default assumptions data.frame
- `hr_after_crossing_from_PH_effect_size()`: Calculate hr after crossing of the hazards from PH effect size
- `cen_rate_from_cen_prop_crossing_hazards()`: calculate censoring rate from censoring proportion
- `true_summary_statistics_crossing_hazards()`: calculate true summary statistics for crossing hazards

Examples

```

one_simulation <- merge(
  assumptions_crossing_hazards(),
  design_fixed_followup(),
  by=NULL
) |>
head(1) |>
generate_crossing_hazards()
head(one_simulation)
tail(one_simulation)
Design <- assumptions_crossing_hazards()
Design
my_design <- merge(
  assumptions_crossing_hazards(),
  design_fixed_followup(),
  by=NULL
)

my_design$final_events <- ceiling((my_design$n_trt + my_design$n_ctrl)*0.75)
my_design$hazard_trt <- NA
my_design <- hr_after_crossing_from_PH_effect_size(my_design, target_power_ph=0.9)
my_design
design <- data.frame(
  crossing = c(2, 4, 6),
  hazard_ctrl = c(0.05, 0.05, 0.05),
  hazard_trt_before = c(0.025, 0.025, 0.025),
  hazard_trt_after = c(0.1, 0.1, 0.1),
  censoring_prop = c(0.1, 0.3, 0.2),
  n_trt = c(50, 50, 50),
  n_ctrl = c(50, 50, 50),
  followup = c(200, 200, 200),
  recruitment = c(50, 50, 50)
)
cen_rate_from_cen_prop_crossing_hazards(design)
my_design <- merge(
  assumptions_crossing_hazards(),
  design_fixed_followup(),
  by=NULL
)
my_design$followup <- 15
my_design <- true_summary_statistics_crossing_hazards(my_design)
my_design

```

generate_delayed_effect

Generate Dataset with delayed effect

Description

Generate Dataset with delayed effect

Create an empty assumptions data.frame for generate_delayed_effect
 Calculate hr after onset of treatment effect
 Calculate true summary statistics for scenarios with delayed treatment effect

Usage

```
generate_delayed_effect(condition, fixed_objects = NULL)

assumptions_delayed_effect(print = interactive())

hr_after_onset_from_PH_effect_size(
  design,
  target_power_ph = NA_real_,
  final_events = NA_real_,
  target_alpha = 0.025
)

cen_rate_from_cen_prop_delayed_effect(design)

true_summary_statistics_delayed_effect(
  Design,
  cutoff_stats = NULL,
  milestones = NULL,
  fixed_objects = NULL
)
```

Arguments

condition	condition row of Design dataset
fixed_objects	additional settings, see details
print	print code to generate parameter set?
design	design data.frame
target_power_ph	target power under proportional hazards
final_events	target events for inversion of Schönfeld Formula defaults to condition\$final_events
target_alpha	target one-sided alpha level for the power calculation
Design	Design data.frame for delayed effect
cutoff_stats	(optionally named) cutoff times, see details
milestones	(optionally named) vector of times at which milestone survival should be calculated

Details

Condition has to contain the following columns:

- n_trt number of patients in treatment arm

- `n_ctrl` number of patients in control arm
- delay time until onset of effect
- `hazard_ctrl` hazard in the control arm = hazard before onset of treatment effect
- `hazard_trt` hazard in the treatment arm after onset of treatment effect

If `fixed_objects` is given and contains an element `t_max`, then this is used as the cutoff for the simulation used internally. If `t_max` is not given in this way the 1-(1/10000) quantile of the survival distribution in the control or treatment arm is used (which ever is larger).

`assumptions_delayed_effect` generates a default design data.frame for use with `generate_delayed_effect`. If `print` is `TRUE` code to produce the template is also printed for copying, pasting and editing by the user. (This is the default when run in an interactive session.)

`hr_after_onset_from_PH_effect_size` calculates the hazard ratio after onset of treatment effect as follows: First, the hazard ratio needed to achieve the desired power under proportional hazards is calculated by inverting Schönfeld's sample size formula. Second the median survival times for both arm under this hazard ratio and proportional hazards are calculated. Finally the hazard rate of the treatment arm after onset of treatment effect is set such that the median survival time is the same as the one calculated under proportional hazards.

This is a heuristic and to some extent arbitrary approach to calculate hazard ratios that correspond to reasonable and realistic scenarios.

`cen_rate_from_cen_prop_delayed_effect` takes the proportion of censored patients from the column `censoring_prop`. This column describes the proportion of patients who are censored randomly before experiencing an event, without regard to administrative censoring.

`cutoff_stats` are the times used to calculate the statistics like average hazard ratios and RMST, that are only calculated up to a certain point.

Value

For `generate_delayed_effect`: A dataset with the columns `t` (time) and `trt` (1=treatment, 0=control), `evt` (event, currently `TRUE` for all observations)

For `assumptions_delayed_effect`: a design tibble with default values invisibly

For `hr_after_onset_from_PH_effect_size`: the design data.frame passed as argument with the additional column `hazard_trt`.

for `cen_rate_from_cen_prop_delayed_effect`: design data.frame with the additional column `random_withdrawal`

For `true_summary_statistics_delayed_effect`: the design data.frame passed as argument with additional columns

Functions

- `generate_delayed_effect()`: simulates a dataset with delayed treatment effect
- `assumptions_delayed_effect()`: generate default assumptions data.frame
- `hr_after_onset_from_PH_effect_size()`: Calculate hr after onset of treatment effect of the hazards from PH effect size
- `cen_rate_from_cen_prop_delayed_effect()`: calculate censoring rate from censoring proportion

- `true_summary_statistics_delayed_effect()`: calculate true summary statistics for delayed effect

Examples

```

one_simulation <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
) |>
head(1) |>
generate_delayed_effect()
head(one_simulation)
tail(one_simulation)
Design <- assumptions_delayed_effect()
Design
my_design <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
)

my_design$hazard_ctrl <- 0.05
my_design$final_events <- ceiling((my_design$n_trt + my_design$n_ctrl)*0.75)
my_design$hazard_trt <- NA
my_design <- hr_after_onset_from_PH_effect_size(my_design, target_power_ph=0.9)
my_design
design <- expand.grid(
  delay=seq(0, 10, by=5),          # delay of 0, 1, ..., 10 days
  hazard_ctrl=0.2,                # hazard under control and before treatment effect
  hazard_trt=0.02,               # hazard after onset of treatment effect
  censoring_prop=c(0.1, 0.25, 0.01), # 10%, 25%, 1% random censoring
  followup=100,                  # followup of 100 days
  n_trt=50,                      # 50 patients treatment
  n_ctrl=50                      # 50 patients control
)
cen_rate_from_cen_prop_delayed_effect(design)
my_design <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
)
my_design <- true_summary_statistics_delayed_effect(my_design)
my_design

```

Description

Generate Dataset with different treatment effect in subgroup
 Create an empty assumptions data.frame for generate_subgroup
 Calculate true summary statistics for scenarios with differential treatment effect in subgroup
 Calculate hazards in treatment arm in subgroup and compliment

Usage

```
generate_subgroup(condition, fixed_objects = NULL)

assumptions_subgroup(print = interactive())

true_summary_statistics_subgroup(
  Design,
  cutoff_stats = NULL,
  milestones = NULL,
  fixed_objects = NULL
)

hazard_subgroup_from_PH_effect_size(
  design,
  target_power_ph = NA_real_,
  final_events = NA_real_,
  target_alpha = 0.025
)

cen_rate_from_cen_prop_subgroup(design)
```

Arguments

condition	condition row of Design dataset
fixed_objects	additional settings, see details
print	print code to generate parameter set?
Design	Design data.frame for subgroup
cutoff_stats	(optionally named) cutoff times, see details
milestones	(optionally named) vector of times at which milestone survival should be calculated
design	design data.frame
target_power_ph	target power under proportional hazards
final_events	target events for inversion of Schönfeld Formula, defaults to condition\$final_events
target_alpha	target one-sided alpha level for the power calculation

Details

Condition has to contain the following columns:

- `n_trt` number of patients in treatment arm
- `n_ctrl` number of patients in control arm
- `hazard_ctrl` hazard in the control arm
- `hazard_trt` hazard in the treatment arm for not cured patients
- `hazard_subgroup` hazard in the subgroup in the treatment arm
- `prevalence` proportion of cured patients

`assumptions_subgroup` generates a default design `data.frame` for use with `generate_subgroup`. If `print` is `TRUE` code to produce the template is also printed for copying, pasting and editing by the user. (This is the default when run in an interactive session.)

`cutoff_stats` are the times used to calculate the statistics like average hazard ratios and RMST, that are only calculated up to a certain point.

`hazard_subgroup_from_PH_effect_size` calculates the hazard rate in the subgroup and the complement of the subgroup in the treatment arm as follows: First, the hazard ratio needed to archive the desired power under proportional hazards is calculated by inverting Schönfeld's sample size formula. Second the median survival times for both arms under this hazard ratio and proportional hazards are calculated. Finally the hazard rate of the treatment arm in the subgroup and its complement are set such that the median survival time is the same as the one calculated under proportional hazards.

This is a heuristic and to some extent arbitrary approach to calculate hazard ratios that correspond to reasonable and realistic scenarios.

`cen_rate_from_cen_prop_subgroup` takes the proportion of censored patients from the column `censoring_prop`. This column describes the proportion of patients who are censored randomly before experiencing an event, without regard to administrative censoring.

Value

For `generate_subgroup`: A dataset with the columns `t` (time) and `trt` (1=treatment, 0=control), `evt` (event, currently `TRUE` for all observations)

For `assumptions_subgroup`: a design tibble with default values invisibly

For `true_summary_statistics_subgroup`: the design `data.frame` passed as argument with the additional columns

For `hazard_subgroup_from_PH_effect_size`: the design `data.frame` passed as argument with the additional columns `hazard_trt` and `hazard_subgroup`.

for `cen_rate_from_cen_prop_subgroup`: design `data.frame` with the additional column `random_withdrawal`

Functions

- `generate_subgroup()`: simulates a dataset with a mixture of cured patients
- `assumptions_subgroup()`: generate default assumptions `data.frame`
- `true_summary_statistics_subgroup()`: calculate true summary statistics for subgroup
- `hazard_subgroup_from_PH_effect_size()`: Calculate hazards in treatment arm
- `cen_rate_from_cen_prop_subgroup()`: calculate censoring rate from censoring proportion

Examples

```

one_simulation <- merge(
  assumptions_subgroup(),
  design_fixed_followup(),
  by=NULL
) |>
head(1) |>
generate_subgroup()
head(one_simulation)
tail(one_simulation)
Design <- assumptions_subgroup()
Design
my_design <- merge(
  assumptions_subgroup(),
  design_fixed_followup(),
  by=NULL
)
my_design <- true_summary_statistics_subgroup(my_design)
my_design

my_design <- merge(
  assumptions_subgroup(),
  design_fixed_followup(),
  by=NULL
)

my_design$hazard_trt <- NA
my_design$hazard_subgroup <- NA
my_design$hr_subgroup_relative <- 0.9
my_design$final_events <- ceiling((my_design$n_ctrl + my_design$n_trt) * 0.75)
my_design <- hazard_subgroup_from_PH_effect_size(my_design, target_power_ph=0.9)
my_design
design <- expand_grid(
  hazard_ctrl=0.2,           # hazard under control and before treatment effect
  hazard_trt=0.02,          # hazard after onset of treatment effect
  hazard_subgroup=0.01,     # hazard in the subgroup in treatment
  prevalence = c(0.2, 0.5),  # subgroup prevalence
  censoring_prop=c(0.1, 0.25, 0.01), # 10%, 25%, 1% random censoring
  followup=100,             # followup of 100 days
  n_trt=50,                  # 50 patients treatment
  n_ctrl=50                  # 50 patients control
)
cen_rate_from_cen_prop_subgroup(design)

```

labs_from_labels

Add ggplot axis labels from labels attribute

Description

Add ggplot axis labels from labels attribute

Usage

```
labs_from_labels(gg)
```

Arguments

`gg` a ggplot object

Value

a ggplot object

Examples

```
library("ggplot2")
test <- mtcars
# add a label attribute
attr(test$cyl, "label") <- "cylinders"

# plot with the variable names as axis titles
gg1 <- ggplot(test, aes(x=wt, y=cyl)) +
  geom_point()
gg1

# add labels where defined in the attribute
gg2 <- ggplot(test, aes(x=wt, y=cyl)) +
  geom_point()

gg2 <- labs_from_labels(gg2)
gg2
```

mixture_haz_fun	<i>Fast implementation of hazard, cumulative hazard, ... for mixtures of subpopulations</i>
-----------------	---

Description

Fast implementation of hazard, cumulative hazard, ... for mixtures of subpopulations

Usage

```
mixture_haz_fun(p, pdfs, survs)
```

```
mixture_cumhaz_fun(p, survs)
```

```
mixture_cdf_fun(p, cdfs)
```

```
mixture_pdf_fun(p, pdfs)
```

```
mixture_surv_fun(p, survs)
```

```
mixture_quant_fun(p, cdfs, quants)
```

```
mixture_rng_fun(p, rngs)
```

Arguments

p	vector of probabilities of the mixture
pdfs	list of probability density functions of the mixture components
survs	list of survival functions of the mixture components
cdfs	list of cumulative density functions of the mixture components
quants	list of quantile functions of the mixture components
rngs	random number generating functions of the components

Details

the last time interval extends to +Inf

mixture_quant_fun relies on numeric root finding and is therefore not as fast as miniPCH::qpch_fun.

mixture_rng samples the counts from the respective mixtures from a multinomial distribution with parameter p and then samples from the components and shuffles the result.

Value

A function with one parameter, a vector of times/probabilities where the function should be evaluated.

Functions

- mixture_haz_fun(): hazard function of mixture
- mixture_cumhaz_fun(): cumulative hazard function of mixture
- mixture_cdf_fun(): cumulative density function of mixture
- mixture_pdf_fun(): probability density function of mixture
- mixture_surv_fun(): survival function of mixture
- mixture_quant_fun(): quantile function of mixture
- mixture_rng_fun(): quantile function of mixture

Examples

```
haz <- mixture_haz_fun(
  p = c(0.3, 0.7),
  pdfs = list(
    miniPCH::dpch_fun(0, 0.1),
    miniPCH::dpch_fun(c(0,5), c(0.1, 0.12))
  ),
```

```

    survs = list(
      miniPCH::spch_fun(0, 0.1),
      miniPCH::spch_fun(c(0,5), c(0.1, 0.12))
    )
  )
plot(haz(seq(0, 30, by=0.15)), ylim=c(0, 0.2), type="l")
abline(h=0)
cumhaz <- mixture_cumhaz_fun(
  p = c(0.3, 0.7),
  survs = list(
    miniPCH::spch_fun(0, 0.1),
    miniPCH::spch_fun(c(0,5), c(0.1, 0.12))
  )
)
plot(cumhaz(seq(0, 30, by=0.15)), type="l")
cdf <- mixture_cdf_fun(
  p = c(0.3, 0.7),
  cdfs = list(
    miniPCH::ppch_fun(0, 0.1),
    miniPCH::ppch_fun(c(0,5), c(0.1, 0.12))
  )
)
plot(cdf(seq(0, 30, by=0.15)), type="l")
pdf <- mixture_pdf_fun(
  p = c(0.3, 0.7),
  pdfs = list(
    miniPCH::dpch_fun(0, 0.1),
    miniPCH::dpch_fun(c(0,5), c(0.1, 0.12))
  )
)
plot(pdf(seq(0, 30, by=0.15)), type="l")
surv <- mixture_surv_fun(
  p = c(0.3, 0.7),
  survs = list(
    miniPCH::spch_fun(0, 0.1),
    miniPCH::spch_fun(c(0,5), c(0.1, 0.12))
  )
)
plot(surv(seq(0, 30, by=0.15)), type="l")

quant <- mixture_quant_fun(
  p = c(0.3, 0.7),
  cdfs = list(
    miniPCH::ppch_fun(0, 0.1),
    miniPCH::ppch_fun(c(0,5), c(0.1, 0.12))
  ),
  quants = list(
    miniPCH::qpch_fun(0, 0.1),
    miniPCH::qpch_fun(c(0,5), c(0.1, 0.12))
  )
)

x <- seq(0, 1, by=0.015)

```

```

plot(x, quant(x), type="l")
rng <- mixture_rng_fun(
  p = c(0.3, 0.7),
  rngs = list(
    miniPCH::rpch_fun(0, 0.1, discrete = TRUE),
    miniPCH::rpch_fun(c(0,5), c(0.1, 0.12), discrete = TRUE)
  )
)
hist(rng(100))

```

progression_cdf_fun	<i>Fast implementation of cumulative density function, survival function, ... for scenarios with progression</i>
---------------------	--

Description

Fast implementation of cumulative density function, survival function, ... for scenarios with progression

Usage

```

progression_cdf_fun(hazard_before, prog_rate, hazard_after)

progression_surv_fun(hazard_before, prog_rate, hazard_after)

progression_pdf_fun(hazard_before, prog_rate, hazard_after)

progression_haz_fun(hazard_before, prog_rate, hazard_after)

progression_quant_fun(hazard_before, prog_rate, hazard_after)

```

Arguments

hazard_before	hazard for death before progression
prog_rate	hazard rate for progression
hazard_after	hazard for death after progression

Details

Calculations are done by viewing the disease process as a three state (non-progressed disease, progressed disease, death) continuous time markov chain. Calculations can then easily be done using the matrix exponential function and Q-matrices.

Value

A function with one parameter, a vector of times/probabilities where the function should be evaluated.

Functions

- progression_cdf_fun(): cumulative density function for progression scenario
- progression_surv_fun(): survival function for progression scenario
- progression_pdf_fun(): probability density function for progression scenario
- progression_haz_fun(): hazard function for progression scenario
- progression_quant_fun(): quantile function for progression scenario

Examples

```
cdf <- progression_cdf_fun(  
  hazard_before = m2r(48),  
  prog_rate = m2r(18),  
  hazard_after = m2r(6)  
)  
t <- 0:1000  
plot(t, cdf(t), type="l")  
surv <- progression_surv_fun(  
  hazard_before = m2r(48),  
  prog_rate = m2r(18),  
  hazard_after = m2r(6)  
)  
t <- 0:1000  
plot(t, surv(t), type="l")  
pdf <- progression_pdf_fun(  
  hazard_before = m2r(48),  
  prog_rate = m2r(18),  
  hazard_after = m2r(6)  
)  
t <- 0:1000  
plot(t, pdf(t), type="l")  
haz <- progression_haz_fun(  
  hazard_before = m2r(48),  
  prog_rate = m2r(18),  
  hazard_after = m2r(6)  
)  
t <- 0:1000  
plot(t, haz(t), type="l")  
quant <- progression_quant_fun(  
  hazard_before = m2r(48),  
  prog_rate = m2r(18),  
  hazard_after = m2r(6)  
)  
p <- seq(0,0.99, by=.01)  
plot(p, quant(p), type="l")
```

r2m	<i>Functions to Convert Between Days and Months and Medians and Rates</i>
-----	---

Description

Some functions to convert between days and months and rates and medians.

Usage

r2m(lambda)

m2r(med)

m2d(mon)

d2m(day)

Arguments

lambda	hazard rate
med	median in months
mon	time in months
day	time in days

Value

median survival time in months (r2m)
hazard rate per day (m2r)
time in days (m2d)
time in months (d2m)

Functions

- r2m(): daily rate to median in months
- m2r(): median to months to daily rate
- m2d(): months to days
- d2m(): days to months

Examples

r2m(0.002)
m2r(12)
m2d(1)
d2m(31)

random_censoring_exp *Apply Random Exponentially Distributed Censoring*

Description

Apply Random Exponentially Distributed Censoring

Usage

```
random_censoring_exp(dat, rate, discrete = TRUE)
```

Arguments

dat	the dataset to apply the random censoring to
rate	time of end of enrollment
discrete	should the censoring times be rounded to whole days?

Value

Returns a Function with one argument dat that modifies a dataset generated by the generate functions by censoring the times and setting the event indicator to FALSE for censored observations.

Examples

```
one_simulation <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
) |>
  head(1) |>
  generate_delayed_effect()

# apply censoring to dataset
censored_sim <- random_censoring_exp(one_simulation, 0.01)

# plot
# uncensored (blue) observations are the same for original and modified
# dataset
# censored (red) observations are smaller than the uncensored ones
plot(
  one_simulation$t,
  censored_sim$t,
  col=ifelse(censored_sim$evt, "blue", "red"),
  xlab = "uncensored times",
  ylab = "censored times"
)
abline(0,1)
```

recruitment_uniform *Add recruitment time to Dataset*

Description

Add recruitment time to Dataset

Apply Administrative Censoring After Fixed Time

Apply Administrative Censoring After Fixed Number of Events

Usage

```
recruitment_uniform(
  dat,
  recruitment_until,
  recruitment_from = 0,
  discrete = TRUE
)

admin_censoring_time(dat, followup, keep_non_recruited = FALSE)

admin_censoring_events(
  dat,
  events,
  keep_non_recruited = FALSE,
  on_incomplete = "ignore"
)
```

Arguments

dat	a simulated dataset
recruitment_until	time of end of recruitment
recruitment_from	time of start of recruitment (defaults to 0)
discrete	should the recruitment time be rounded to full days?
followup	followup time
keep_non_recruited	should patients recruited after end of study be kept
events	number of events after which the dataset is analyzed
on_incomplete	what to do if there are fewer events than planned "ignore", "warn", "stop"

Details

The Dataset has to include a column `rec_time` containing the recruitment time as well as the columns with the event times `t` and a column with the event indicator `evt`.

Times and event indicators for patients recruited after followup are set to NA.

The Dataset has to include a column `rec_time` containing the recruitment time as well as the columns with the event times `t` and a column with the event indicator `evt`.

Times and event indicators for patients recruited after followup are set to NA.

If there are less events than planned for study end `on_incomplete` defines what should be done. "ignore" simply returns the dataset with the maximum of the observed times as followup. "warn" does the same but gives a warning. "stop" stops with an error.

Value

Returns the dataset with added recruitment times.

Returns the dataset with administrative censoring after followup, adds the attribute `followup` with the followup time to the dataset.

Returns the dataset with administrative censoring after events events, adds the attribute `followup` with the followup time to the dataset.

Functions

- `recruitment_uniform()`: add recruitment time
- `admin_censoring_time()`: apply administrative censoring after fixed time
- `admin_censoring_events()`: apply administrative censoring after fixed number of events

Examples

```
dat <- data.frame(t=c(0, 1, 2), trt=c(FALSE, FALSE, TRUE))
recruitment_uniform(dat, 7, 0)
dat <- data.frame(
  t = 1:10,
  rec_time = rep(1:5, each=2),
  trt = rep(c(TRUE, FALSE), times=5),
  evt = rep(TRUE, times=10)
)
dat

admin_censoring_time(dat, 4)
admin_censoring_time(dat, 4, keep_non_recruited = TRUE)

dat_censored <- admin_censoring_time(dat, 5)
attr(dat_censored, "followup")
dat <- data.frame(
  t = 1:10,
  rec_time = rep(2*(1:5), each=2),
  trt = rep(c(TRUE, FALSE), times=5),
  evt = rep(TRUE, times=10)
)
```

```

dat

admin_censoring_events(dat, 4)
admin_censoring_events(dat, 4, keep_non_recruited = TRUE)

dat_censored <- admin_censoring_events(dat, 4)
attr(dat_censored, "followup")

```

rename_results_column *Rename Columns in Simulation Results and Update Attributes*

Description

Rename Columns in Simulation Results and Update Attributes

Usage

```

rename_results_column(results, rename)

rename_results_column_pattern(results, pattern, replacement)

```

Arguments

results	SimDesign object
rename	named vector of new names
pattern	regexp pattern as understood by <code>stringr::str_replace_all</code>
replacement	replacement as understood by <code>stringr::str_replace_all</code>

Value

SimDesign object with updated column names

Functions

- `rename_results_column()`: Rename Columns in Simulation Results
- `rename_results_column_pattern()`: Rename Columns in Simulation Results by Pattern

Examples

```

condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
) |>
  tail(4) |>
  true_summary_statistics_delayed_effect(cutoff_stats = 15)

sim_results <- runSimulation(

```

```

    design=condition,
    replications=10,
    generate=generate_delayed_effect,
    analyse=list(
      logrank = analyse_logrank(alternative = "one.sided"),
      mwlrt = analyse_modelstly_weighted(t_star = m2d(24))
    ),
    summarise = create_summarise_function(
      logrank = summarise_test(0.025),
      mwlrt = summarise_test(0.025)
    )
  )
)

names(sim_results)
attr(sim_results, "design_names")

sim_results <- sim_results |>
  rename_results_column(c("delay"="onset"))

names(sim_results)
attr(sim_results, "design_names")

condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
) |>
  tail(4) |>
  true_summary_statistics_delayed_effect(cutoff_stats = 15)

sim_results <- runSimulation(
  design=condition,
  replications=10,
  generate=generate_delayed_effect,
  analyse=list(
    logrank = analyse_logrank(alternative = "one.sided"),
    mwlrt = analyse_modelstly_weighted(t_star = m2d(24))
  ),
  summarise = create_summarise_function(
    logrank = summarise_test(0.025),
    mwlrt = summarise_test(0.025)
  )
)

names(sim_results)
attr(sim_results, "design_names")

sim_results <- sim_results |>
  rename_results_column_pattern(pattern = "_0.025", replacement = "")

names(sim_results)
attr(sim_results, "design_names")

```

results_pivot_longer *Functions for Plotting and Reporting Results*

Description

Functions for Plotting and Reporting Results

Usage

```
results_pivot_longer(data, exclude_from_methods = c("descriptive"))

combined_plot(
  data,
  methods,
  xvars,
  yvar,
  facet_x_vars = c(),
  facet_y_vars = c(),
  split_var = 1,
  heights_plots = c(3, 1),
  scale_stairs = NULL,
  grid_level = 2,
  scales = "fixed",
  hlines = numeric(0),
  use_colours = NULL,
  use_shapes = NULL,
  expand_x_axis = c(0.05, 0, 0.05, 0)
)
```

Arguments

data	for results_pivot_longer: simulation result as returned by SimDesign, for combined_plot: simulation results in long format, as returned by results_pivot_longer.
exclude_from_methods	"methods" that should not be pivoted into long format
methods	methods to include in the plot
xvars	ordered vector of variable names to display on the x axis
yvar	variable name of the variable to be displayed on the y axis (metric)
facet_x_vars	vector of variable names to create columns of facets
facet_y_vars	vector of variable names to create rows of facets
split_var	where should the lines be split, see details
heights_plots	relative heights of the main plot and the stairs on the bottom

scale_stairs	this argument is deprecated and will be ignored
grid_level	depth of loops for which the grid-lines are drawn
scales	passed on to facet_grid
hlines	position of horizontal lines, passed as yintercept to geom_hline
use_colours	optional named vector of colours used in scale_colour_manual
use_shapes	optional named vector of shapes used in scale_shape_manual
expand_x_axis	axis expansion factor, passed to scale_x_continuous

Details

With `exclude_from_methods` descriptive statistics or results of reference methods can be kept as own columns and used like the columns of the simulation parameters.

`use_colours` and `use_shapes` both use the method variable in their respective aesthetics.

`split_var` break the lines after the 1st, 2nd, ... variable in `xvars`. Use 0 for one continuous line per method.

Value

dataset in long format with one row per method and scenario and one column per metric

a `ggplot/patchwork` object containing the plots

Functions

- `results_pivot_longer()`: pivot simulation results into long format
- `combined_plot()`: Nested Loop Plot with optional Facets

Examples

```
data("combination_tests_delayed")

combination_tests_delayed |>
  results_pivot_longer() |>
  head()

library("ggplot2")
library("patchwork")
data("combination_tests_delayed")

results_long <- results_pivot_longer(combination_tests_delayed)

# plot the rejection rate of two methods
combined_plot(
  results_long,
  c("logrank", "mwlrt", "maxcombo"),
  c("hr", "n_pat_design", "delay", "hazard_ctrl", "recruitment"),
  "rejection_0.025",
  grid_level=2
```

```

)

# use custom colour and shape scales
# this can be used to group methods by shape or colour
# this is also helpful if methods should have the same aesthetics across plots
my_colours <- c(
  logrank="black",
  mwlrt="blue",
  maxcombo="green"
)

my_shapes <- c(
  logrank=1,
  mwlrt=2,
  maxcombo=2
)

combined_plot(
  results_long,
  c("logrank", "mwlrt", "maxcombo"),
  c("hr", "n_pat_design", "delay", "hazard_ctrl", "recruitment"),
  "rejection_0.025",
  grid_level=2,
  use_colours = my_colours,
  use_shapes = my_shapes
)

# if one has a dataset of metadata with categories of methods
# one could use those two definitions
# colours for methods, same shapes for methods of same category
metadata <- data.frame(
  method = c("logrank", "mwlrt", "maxcombo"),
  method_name = c("logrank test", "modestly weighed logrank test", "maxcombo test"),
  category = c("logrank test", "combination test", "combination test")
)

my_colours <- rainbow(n=nrow(metadata)) |>
  sample() |>
  setNames(metadata$method)

my_shapes <- metadata$category |>
  as.factor() |>
  as.integer() |>
  setNames(metadata$method)

combined_plot(
  results_long,
  c("logrank", "mwlrt", "maxcombo"),
  c("hr", "n_pat_design", "delay", "hazard_ctrl", "recruitment"),
  "rejection_0.025",
  grid_level=2,
  use_colours = my_colours,
  use_shapes = my_shapes

```


)

shhr_gg

Plot of survival, hazard and hazard ratio of two groups as a function of time using ggplot and patchwork

Description

Plot of survival, hazard and hazard ratio of two groups as a function of time using ggplot and patchwork

Usage

```
shhr_gg(
  A,
  B,
  main = NULL,
  sub = NULL,
  group_names = c("control", "treatment"),
  lab_time = "Days",
  lab_group = "Group",
  trafo_time = identity,
  colours = palette()[c(1, 3)],
  linetypes = c(1, 3),
  linewidths = c(1.3, 1.3),
  as_list = FALSE
)
```

Arguments

A	mixpch object for group 1 (reference)
B	mixpch object for group 2
main	Title for the overall plot
sub	Subtitle for the overall plot
group_names	Group Names
lab_time	Title for the time axis
lab_group	Title group legend
trafo_time	Function to transform time
colours	vector of two colours
linetypes	vector of two linetypes
linewidths	vector of two linewidths
as_list	return a list of ggplot objects instead of a patchwork object

Value

a patchwork object as defined in the patchwork package or a list of ggplot objects if as_list=TRUE.

Examples

```
library(ggplot2)
library(patchwork)
library(nph)
B <- pchaz(c(0, 10, 100), c(0.1, 0.05))
A <- pchaz(c(0, 100), c(0.1))
shhr_gg(A, B)
shhr_gg(A, B, lab_time="Months", trafo_time=d2m)
```

summarise_estimator	<i>Generic Summarise function for esitmators</i>
---------------------	--

Description

Generic Summarise function for esitmators

Usage

```
summarise_estimator(
  est,
  real,
  lower = NULL,
  upper = NULL,
  null = NULL,
  est_sd = NULL,
  name = NULL
)
```

Arguments

est	estimator, expression evaluated in results
real	real summary statistic, expression evaluated in condition
lower	lower CI, expression evaluated in results
upper	upper CI, expression evaluated in results
null	parameter value under the null hypothesis
est_sd	standard deviation estimated by the method, evaluated in results
name	name for the summarise function, appended to the name of the analysis method in the final results

Details

The different parameters are evaluated in different environments, `est`, `lower`, `upper`, `est_sd` refer to output of the method and are evaluated in the results dataset. `real` refers to a real value of a summary statistic in this scenario and is therefore evaluated in the condition dataset. `null` and `name` are constants and directly evaluated when the function is defined. The argument `null`, the parameter value under the null hypothesis is used to output the rejection rate based on the confidence interval. Which is output in the column `null_cover`

Value

A function that can be used in Summarise that returns a data frame with summary statistics of the performance measures in the columns.

Examples

```
# generate the design matrix and append the true summary statistics
condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
) |>
tail(4) |>
head(1) |>
true_summary_statistics_delayed_effect(cutoff_stats = 15)

# create some summarise functions
summarise_all <- create_summarise_function(
  coxph=summarise_estimator(hr, gAHR_15, hr_lower, hr_upper, name="gAHR"),
  coxph=summarise_estimator(hr, hazard_trt/hazard_ctrl, hr_lower, hr_upper, name="HR"),
  coxph=summarise_estimator(hr, NA_real_, name="NA")
)

# runs simulations
sim_results <- runSimulation(
  design=condition,
  replications=10,
  generate=generate_delayed_effect,
  analyse=list(
    coxph=analyse_coxph()
  ),
  summarise = summarise_all
)

# mse is missing for the summarise function in which the real value was NA
sim_results[, names(sim_results) |> grepl(pattern="\\.mse$")]
# but the standard deviation can be estimated in all cases
sim_results[, names(sim_results) |> grepl(pattern="\\.sd_est$")]
```

summarise_test	<i>Generic summarise function for tests</i>
----------------	---

Description

Generic summarise function for tests

Usage

```
summarise_test(alpha, name = NULL)
```

Arguments

alpha	the significance level(s)
name	name for the summarise function, appended to the name of the analysis method in the final results

Value

A function that can be used in Summarise that returns a data frame with the columns

- rejection_X
- rejection_Y
- ...

Where X, Y, ... are the alpha levels given in the argument

Examples

```
condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
) |>
  tail(4) |>
  head(1)

summarise_all <- create_summarise_function(
  logrank=summarise_test(alpha=c(0.5, 0.9, 0.95, 0.99))
)

# runs simulations
sim_results <- runSimulation(
  design=condition,
  replications=100,
  generate=generate_delayed_effect,
  analyse=list(
    logrank=analyse_logrank()
  ),
)
```

```

    summarise = summarise_all
  )

sim_results[, grepl("rejection", names(sim_results))]

```

upsert_merge	<i>Merge results from additional or updated simulations</i>
--------------	---

Description

Merge results from additional or updated simulations

Usage

```

upsert_merge(x, y, by)

merge_additional_results(
  old,
  new,
  design_names = NULL,
  descriptive_regex = NULL
)

```

Arguments

x	left data.frame
y	right data.frame
by	columns to match by
old	old results
new	new/additional results
design_names	names of the parameterst
descriptive_regex	regular expression for columns of descriptive statistics

Details

updates columns in x with values from matched rows in y and add joins columns from y not present in x. Calls `rows_upsert` and then `full_join`.

if `design_names` is omitted its value is taken from the `design_names` attribute of the simulation results.

If `descriptive_regex` is given, columns matching the regular expression in both datasets are compared, a warning is given, if the values of those columns do not match. This is intended to compare descriptive statistics or results of unchanged analysis methods to ensure, that both results stem from an exact replication of the simulation results.

Value

a data.frame

a data.frame of the merged simulation results

Functions

- `upsert_merge()`: Update or add Rows and Columns

Examples

```
a <- data.frame(x=5:2, y=5:2, a=5:2)
b <- data.frame(x=1:4, y=1:4+10, b=1:4*10)
upsert_merge(a, b, by="x")

condition <- merge(
  assumptions_delayed_effect(),
  design_fixed_followup(),
  by=NULL
) |>
tail(4) |>
true_summary_statistics_delayed_effect(cutoff_stats = 15)

condition_1 <- condition[1:2, ]
condition_2 <- condition[3:4, ]

# runs simulations
sim_results_1 <- runSimulation(
  design=condition_1,
  replications=100,
  generate=generate_delayed_effect,
  analyse=list(
    logrank = analyse_logrank(alternative = "one.sided"),
    maxcombo = analyse_logrank(alternative = "one.sided")
  ),
  summarise = create_summarise_function(
    logrank = summarise_test(0.025),
    maxcombo = summarise_test(0.025)
  )
)

sim_results_2 <- runSimulation(
  design=condition_2,
  replications=100,
  generate=generate_delayed_effect,
  analyse=list(
    logrank = analyse_logrank(alternative = "one.sided"),
    maxcombo = analyse_logrank(alternative = "one.sided")
  ),
  summarise = create_summarise_function(
    logrank = summarise_test(0.025),
    maxcombo = summarise_test(0.025)
  )
)
```

```

)

sim_results_3 <- runSimulation(
  design=condition,
  replications=100,
  generate=generate_delayed_effect,
  analyse=list(
    mwlrt = analyse_modelstly_weighted(t_star = m2d(24))
  ),
  summarise = create_summarise_function(
    mwlrt = summarise_test(0.025)
  )
)

all_results <- sim_results_1 |>
  merge_additional_results(sim_results_2) |>
  merge_additional_results(sim_results_3)

all_results |>
  subset(select=c(delay, logrank.rejection_0.025, maxcombo.rejection_0.025, mwlrt.rejection_0.025))

```

wrap_all_in_trycatch *Wrappers around Analyse Functions*

Description

Wrappers around Analyse Functions

Usage

```

wrap_all_in_trycatch(
  list_of_functions,
  error = function(e) {
    warning(e$message)
    NA
  }
)

wrap_all_in_preserve_seed(list_of_functions)

```

Arguments

list_of_functions	the list of functions to be wrapped
error	the error function in the tryCatch call

Details

SimDesign redraws data if one analysis function fails. This is not only highly inefficient for large studies, but failure of a method is informative and might be of interest. Moreover redrawing of data might introduce bias if the failure of the method is not independent of the parameter value, which would be a strong assumption.

To avoid redrawing data, we can catch all errors the analysis methods could throw and return NA instead.

This is handled well by the summarise functions generated with `create_summarise_function` other summarise functions might throw errors when trying to `rbind` a data.frame to a scalar NA value. In this case add another error argument. For example `\(e){NULL}` could work in some cases, in other cases you'll have to give a function that returns a data.frame with the same columns as the analyse functions and only NA values.

Analysis functions might use random numbers. If simulations should be replicated this can interfere with the RNG state of other analysis functions. To avoid this you can wrap all analysis function in a `withr::with_preserve_seed` call, so that the RNG state is reset after each analysis function is called. This way adding, removing or changing one analysis function has no effect on the other analysis functions, even if the analysis functions use random numbers.

Value

a list of functions

Functions

- `wrap_all_in_trycatch()`: Wrap all functions in a list in tryCatch calls
- `wrap_all_in_preserve_seed()`: wrap all functions in `withr::with_preserve_seed`

Examples

```
funs1 <- list(\(){stop("test")}, \(){1})
funs2 <- wrap_all_in_trycatch(funs1)
try(lapply(funs1, \(){f()}))
try(lapply(funs2, \(){f()}))

funs1 <- list(\(){rnorm(1)})
funs2 <- list(\(){runif(1)}, \(){rnorm(1)})
funs3 <- funs2 |> wrap_all_in_preserve_seed()
set.seed(1)
lapply(funs1, \(){f()})
set.seed(1)
lapply(funs2, \(){f()})
set.seed(1)
lapply(funs3, \(){f()})
```


Index

* datasets

combination_tests_delayed, 22

admin_censoring_events

(recruitment_uniform), 42

admin_censoring_time

(recruitment_uniform), 42

analyse_aft, 3

analyse_ahr, 4

analyse_coxph, 5

analyse_describe, 6

analyse_diff_median_survival, 7

analyse_gehan_wilcoxon, 9

analyse_group_sequential, 9

analyse_logrank, 11

analyse_logrank_fh_weights, 12

analyse_maxcombo, 13

analyse_milestone_survival, 13

analyse_modelstly_weighted, 15

analyse_piecewise_exponential, 16

analyse_rmst_diff, 17

analyse_weibull, 18

assumptions_crossing_hazards

(generate_crossing_hazards), 25

assumptions_delayed_effect

(generate_delayed_effect), 28

assumptions_progression, 19

assumptions_subgroup

(generate_subgroup), 31

cen_rate_from_cen_prop_crossing_hazards

(generate_crossing_hazards), 25

cen_rate_from_cen_prop_delayed_effect

(generate_delayed_effect), 28

cen_rate_from_cen_prop_progression

(assumptions_progression), 19

cen_rate_from_cen_prop_subgroup

(generate_subgroup), 31

combination_tests_delayed, 22

combined_plot(results_pivot_longer), 46

create_summarise_function, 23

d2m(r2m), 40

design_fixed_followup, 24

design_group_sequential, 25

generate_crossing_hazards, 25

generate_delayed_effect, 28

generate_progression

(assumptions_progression), 19

generate_subgroup, 31

hazard_before_progression_from_PH_effect_size

(assumptions_progression), 19

hazard_subgroup_from_PH_effect_size

(generate_subgroup), 31

hr_after_crossing_from_PH_effect_size

(generate_crossing_hazards), 25

hr_after_onset_from_PH_effect_size

(generate_delayed_effect), 28

labs_from_labels, 34

m2d(r2m), 40

m2r(r2m), 40

merge_additional_results

(upsert_merge), 53

mixture_cdf_fun(mixture_haz_fun), 35

mixture_cumhaz_fun(mixture_haz_fun), 35

mixture_haz_fun, 35

mixture_pdf_fun(mixture_haz_fun), 35

mixture_quant_fun(mixture_haz_fun), 35

mixture_rng_fun(mixture_haz_fun), 35

mixture_surv_fun(mixture_haz_fun), 35

nph::nphparams, 5, 8, 14, 17

progression_cdf_fun, 38

progression_haz_fun

(progression_cdf_fun), 38

- progression_pdf_fun
 - (progression_cdf_fun), 38
- progression_quant_fun
 - (progression_cdf_fun), 38
- progression_rate_from_progression_prop
 - (assumptions_progression), 19
- progression_surv_fun
 - (progression_cdf_fun), 38
- r2m, 40
- random_censoring_exp, 41
- recruitment_uniform, 42
- rename_results_column, 44
- rename_results_column_pattern
 - (rename_results_column), 44
- results_pivot_longer, 46
- shhr_gg, 49
- summarise_describe(analyse_describe), 6
- summarise_estimator, 50
- summarise_group_sequential
 - (analyse_group_sequential), 9
- summarise_test, 52
- true_summary_statistics_crossing_hazards
 - (generate_crossing_hazards), 25
- true_summary_statistics_delayed_effect
 - (generate_delayed_effect), 28
- true_summary_statistics_progression
 - (assumptions_progression), 19
- true_summary_statistics_subgroup
 - (generate_subgroup), 31
- upsert_merge, 53
- wrap_all_in_preserve_seed
 - (wrap_all_in_trycatch), 55
- wrap_all_in_trycatch, 55