

Package ‘FastSurvival’

October 8, 2026

Type Package

Title Fast Survival Analysis and Simulation for Clinical Trials

Version 1.1.0

Description Provides fast alternatives to standard survival analysis functions in the 'survival' package, together with tools for time-to-event trial simulation and sequential analysis. The estimation and testing functions cover a single-time-point Kaplan-Meier estimator (`survfit_fast()`), log-rank tests including weighted and stratified variants (`survdif_fast()`), a closed-form hazard ratio estimator based on the Pike-Halley Estimator method (`coxph_fast()`), restricted mean survival time (`rmst_fast()`), window mean survival time (`wmst_fast()`), milestone survival comparison (`milestone_fast()`), median survival time (`medsurv_fast()`), the max-combo test (`maxcombo_fast()`), the robust modestly-weighted log-rank test (`rmw_fast()`), the weighted Kaplan-Meier (Pepe-Fleming) test (`wkm_fast()`), the average hazard with survival weight (`ahsw_fast()`), and the Kalbfleisch-Prentice average hazard ratio (`ahr_fast()`). The simulation layer generates individual patient data (`simdata_fast()`), determines the calendar time of each analysis from combined event and calendar-time rules (`cutoff_fast()`), performs interim or sequential analyses (`analysis_fast()`), compares several experimental arms with a shared control (`pairwise_fast()`), applies treatment switching at progression or after an interim analysis (`switch_fast()`), and aggregates operating characteristics (`simsummary_fast()`). A visualization layer assembles design-stage scenarios (`gen_scenario_fast()`) and builds analysis-stage Kaplan-Meier curves (`kmcurve_fast()`), each with plot and print methods. All functions are designed for repeated evaluation inside large simulation loops, such as adaptive sample-size re-estimation, probability-of-success calculations, and regional consistency evaluation in multi-regional trials. Core computations are implemented in 'C++' via 'Rcpp' for maximum performance. Methodological background is described in Collett (2014, ISBN:9780429196294).

License MIT + file LICENSE

URL <https://github.com/gosukehommaEX/FastSurvival>,
<https://gosukehommaEX.github.io/FastSurvival/>

BugReports <https://github.com/gosukehommaEX/FastSurvival/issues>

Encoding UTF-8

Language en-US

Depends R (>= 4.1.0)

Imports graphics, grDevices, stats, dqrng, Rcpp, mvtnorm

LinkingTo Rcpp, dqrng

Suggests survival, survRM2, survAH, simtrial, gsDesign, neph, nephRCT, rpsftm, microbenchmark, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

VignetteBuilder knitr

NeedsCompilation yes

Author Gosuke Homma [aut, cre]

Maintainer Gosuke Homma <my.name.is.gosuke@gmail.com>

Repository CRAN

Date/Publication 2026-10-08 16:40:08 UTC

Contents

ahr_fast	3
ahsw_fast	5
analysis_fast	7
coxph_fast	14
cutoff_fast	17
gen_scenario_fast	20
kmcurve_fast	22
maxcombo_fast	23
medsurv_fast	26
milestone_fast	28
pairwise_fast	30
plot.kmcurve_fast	32
plot.scenario_fast	35
print.ahr_fast	36
print.ahsw_fast	37
print.coxph_fast	38
print.kmcurve_fast	38
print.maxcombo_fast	39
print.medsurv_fast	40
print.milestone_fast	41
print.rmst_fast	42
print.rmw_fast	43
print.scenario_fast	43
print.simsummary_fast	44

print.survdiff_fast	45
print.survfit_fast	46
print.wkm_fast	47
print.wmst_fast	48
rmst_fast	49
rmw_fast	51
simdata_fast	54
simsummary_fast	61
survdiff_fast	65
survfit_fast	69
switch_fast	71
wkm_fast	74
wmst_fast	76

Index	79
--------------	-----------

ahr_fast	<i>Fast Kalbfleisch-Prentice average hazard ratio for two groups</i>
----------	--

Description

Estimates the average hazard ratio of Kalbfleisch and Prentice (1981) between two groups over the time interval from 0 to tau, based on the Kaplan-Meier estimator of each group's survival function. This is the Kaplan-Meier (unweighted) special case of the estimator implemented in the archived AHR package, restricted to two groups and recoded in C++ for use inside simulation loops.

Usage

```
ahr_fast(
  time,
  event,
  group,
  control,
  side = 2,
  conf.level = 0.95,
  tau = NULL,
  null.ahr = 1,
  presorted = FALSE
)
```

Arguments

time	vector of right-censored event times
event	0/1 (or logical) event indicators, 1 for an event
group	vector with exactly two distinct values identifying the groups.
control	the value of group that denotes the reference (control) group. The average hazard ratio is reported for the other group relative to it.

side	1 for a one-sided test in the direction of treatment benefit (average hazard ratio below null.ahr) or 2 for a two-sided test (default 2).
conf.level	confidence level for the confidence interval (default 0.95)
tau	upper limit of the interval over which the average hazard ratio is computed. If NULL (default) the largest time observed in both groups is used. A supplied value larger than that gives a warning, because the Kaplan-Meier curves are not estimated beyond it.
null.ahr	value of the average hazard ratio under the null hypothesis used for the Z statistic and p-value (default 1)
presorted	if TRUE, assume time is already sorted in ascending order so that each group's observations are also ascending; this skips the internal sort (default FALSE) The order is checked, and an error is given when it does not hold.

Details

The estimator works with the group shares of the total hazard. Writing S_1 and S_2 for the two survival functions, the reference-group share is $\theta_1 = -\int_0^{\tau} S_2 dS_1 / (1 - S_1(\tau) S_2(\tau))$ over $[0, \tau]$, the comparison-group share is $\theta_2 = 1 - \theta_1$, and the average hazard ratio is $ahr = \theta_2 / \theta_1$. Under proportional hazards with hazard ratio ψ , ahr estimates ψ . A value above 1 indicates higher hazard (worse survival) in the comparison group. The variance of θ_1 is the direct Greenwood-based estimator. The primary test is on the θ (group-share) scale, as in Kalbfleisch and Prentice (1981) and Dormuth et al. (2024, eq. 5): the comparison-group share is compared with its null value (0.5 when $null.ahr = 1$). An equivalent test and a confidence interval on the $\log(ahr)$ scale are also reported.

The integral uses the right-continuous Kaplan-Meier estimate S_2 at each jump of S_1 , exactly as the AHR package does, and θ_2 is obtained as $1 - \theta_1$ rather than by a separate integral. When both groups have events at the same time (tied event times, as with data recorded in whole days or months), the two shares computed this way are not symmetric in the groups, so exchanging `control` does not give exactly the reciprocal average hazard ratio. With continuous times there are no such ties and the estimate is symmetric. Specify `control` as the actual reference group so that the result matches the AHR package with that group as the reference.

This is distinct from `ahsw_fast`, which estimates the Uno-Horiguchi average hazard with survival weight.

Value

An object of class "ahr_fast", a list with elements `ahr` (the average hazard ratio, comparison vs reference), `log.ahr`, `se.loghr`, `lower`, `upper`, `conf.level`, `z` and `p.value` (the primary test on the θ / group-share scale, as in Dormuth et al. 2024 eq. 5), `z.loghr` and `p.value.loghr` (the equivalent test on the $\log(ahr)$ scale), `se.theta` (standard error of the tested comparison-group share), `null.share`, `null.ahr`, `theta` (the two group shares), `var.theta1`, `var.theta2`, `side`, `tau`, `n` (the two group sizes) and `groups`.

References

Kalbfleisch, J. D., & Prentice, R. L. (1981). Estimation of the average hazard ratio. *Biometrika*, 68(1), 105-112.

Dormuth, I., Pauly, M., Rauch, G., & Herrmann, C. (2024). Sample size calculation under nonproportional hazards using average hazard ratios. *Biometrical Journal*, 66(6), e202300271.

See Also

[ahsw_fast](#), [coxph_fast](#)

Examples

```
set.seed(1)
n <- 200
time1 <- rexp(n, 0.1)
time2 <- rexp(n, 0.18)
cens <- rexp(2 * n, 0.05)
obs <- pmin(c(time1, time2), cens)
event <- as.integer(c(time1, time2) <= cens)
group <- rep(c(0, 1), each = n)
ahr_fast(obs, event, group, control = 0, tau = 8)
```

ahsw_fast

Fast Average Hazard with Survival Weight (Two-Group Comparison)

Description

Computes the average hazard with survival weight (AHSW) of Uno and Horiguchi for two groups and the between-group contrasts. The average hazard on the window from 0 to tau is the ratio of the cumulative event probability at tau to the restricted mean survival time at tau, both based on the Kaplan-Meier estimate. The function returns the per-group average hazard, the ratio of average hazards (RAH, treatment over control) on the log scale, and the difference of average hazards (DAH, treatment minus control) on the identity scale, each with a confidence interval and a test (one- or two-sided, following side). The C++ backend walks the pooled sorted data once per group, so the function is suitable for simulation loops with presorted = TRUE.

Usage

```
ahsw_fast(
  time,
  event,
  group,
  control,
  side = 2,
  conf.level = 0.95,
  tau,
  presorted = FALSE
)
```

Arguments

<code>time</code>	A numeric vector of follow-up times for all subjects.
<code>event</code>	An integer or numeric vector of event indicators (1 = event, 0 = censored), aligned with <code>time</code> .
<code>group</code>	A vector of group labels aligned with <code>time</code> .
<code>control</code>	A scalar value indicating which level of group represents the control group.
<code>side</code>	1 for a one-sided test in the direction of treatment benefit (lower average hazard in the treatment group, i.e. a ratio below 1 and a negative difference) or 2 for a two-sided test (default 2). The reported p-values follow this choice; the confidence intervals are always two-sided at <code>conf.level</code> .
<code>conf.level</code>	A single numeric value in (0, 1) specifying the confidence level. Defaults to 0.95.
<code>tau</code>	A single positive numeric value, the truncation time point for the average hazard. A warning is given when it exceeds the largest observed time of either group, because the Kaplan-Meier curve is not estimated beyond that time.
<code>presorted</code>	A logical value. If TRUE, <code>time</code> , <code>event</code> , and <code>group</code> are assumed to be sorted in ascending order of time, and the <code>internal order()</code> call is skipped. If FALSE (default), sorting is handled internally. The order is checked, and an error is given when it does not hold.

Details

The average hazard with survival weight is

$$AH(\tau) = (1 - S(\tau)) / \text{integral over } [0, \tau] \text{ of } S(u) \, du,$$

the ratio of the cumulative event probability at τ to the restricted mean survival time at τ . It can be read as a general censoring-free incidence rate on the window from 0 to τ and stays interpretable under non-proportional hazards. This is a different quantity from the average hazard ratio of Kalbfleisch, which averages the time-varying ratio of hazards rather than forming a single average hazard per group and then contrasting.

Writing the treatment and control average hazards as a_1 and a_0 , the ratio contrast is $RAH = a_1 / a_0$, formed on the log scale with variance $v_{Q1} / n_1 + v_{Q0} / n_0$, and the difference contrast is $DAH = a_1 - a_0$, with variance $v_{U1} / n_1 + v_{U0} / n_0$, using the independence of the two groups. The per-group variance terms v_Q (log scale) and v_U (identity scale) follow the asymptotic variance of Uno and Horiguchi, computed from the Nelson-Aalen increments, the running restricted mean survival time and the at-risk fraction. The confidence interval for RAH is exponentiated from the log scale, and the p-values are based on the normal approximation.

When `presorted = TRUE`, the inputs are assumed to be sorted in ascending order of time, so the `internal order()` call is skipped. Splitting into groups preserves the ascending order within each group.

Value

An object of class "ahsw_fast", a named numeric vector containing the per-group average hazards (`ah.ctrl`, `ah.trt`), the ratio contrast (`rah`, `rah.lower`, `rah.upper`, `p.rah`), and the difference contrast (`dah`, `dah.lower`, `dah.upper`, `p.dah`). The truncation time and confidence level are stored

as attributes `tau` and `conf.level`, and the control label is also stored. Returns NA values (still with class "ahsw_fast") when either group has no events up to `tau` (a zero average hazard) or a non-finite variance.

References

Uno, H., & Horiguchi, M. (2023). Ratio and difference of average hazard with survival weight: new measures to quantify survival benefit of new therapy. *Statistics in Medicine*, 42(7), 936-952.

See Also

[rmst_fast](#) for the restricted mean survival time, and [survfit_fast](#) for the Kaplan-Meier estimate at a time point.

Examples

```
library(survival)

# Average hazard contrasts on the ovarian data
ahsw_fast(ovarian$futime, ovarian$fustat, ovarian$rx, control = 1, tau = 600)

# presorted = TRUE: sort once outside, reuse inside a loop
ord <- order(ovarian$futime)
ahsw_fast(ovarian$futime[ord], ovarian$fustat[ord], ovarian$rx[ord],
          control = 1, tau = 600, presorted = TRUE)

# Validation against survAH
if (requireNamespace("survAH", quietly = TRUE)) {
  arm <- as.numeric(ovarian$rx == 2)
  survAH::ah2(time = ovarian$futime, status = ovarian$fustat,
              arm = arm, tau = 600)
}
```

Description

Performs interim or sequential analyses of simulated two-group time-to-event data at one or more analysis times ("looks"). Each look is defined by a target cumulative number of events (information-based timing), by a calendar time (calendar-based timing), or by a calendar cutoff given separately for every simulated trial (for example by [cutoff_fast](#)). At every look the data are administratively censored at the corresponding calendar cutoff, and the requested statistics are computed for each simulated trial by reusing [survdiff_fast](#), [coxph_fast](#), [rmst_fast](#), [survfit_fast](#), [maxcombo_fast](#), [ahsw_fast](#), [milestone_fast](#), [rmw_fast](#), [ahr_fast](#), [medsurv_fast](#), [wkm_fast](#),

and `wmst_fast`. Optionally the same statistics are also reported within each subgroup. The censoring and time sorting are handled by a C++ backend, and the analysis cores are called with `presorted = TRUE`, so each look avoids a redundant sort.

Usage

```
analysis_fast(
  data,
  control,
  event.looks = NULL,
  time.looks = NULL,
  stat = "logrank",
  tau = NULL,
  t.eval = NULL,
  conf.level = 0.95,
  side = 2,
  by.subgroup = FALSE,
  weight = c("logrank", "fh", "mwlrt", "gehan", "tarone-ware"),
  rho = 0,
  gamma = 0,
  t_star = NULL,
  strata = NULL,
  ms.method = c("wald", "loglog", "mover"),
  s_star = 0.5,
  mc.rho = c(0, 0, 1, 1),
  mc.gamma = c(0, 1, 0, 1),
  abseps = 1e-05,
  maxpts = 25000,
  medsurv.method = c("km", "nph"),
  medsurv.bw = NULL,
  wkm.weight = c("PF", "sqrtPF", "constant"),
  wmst.tau1 = 0,
  wmst.tau2 = NULL,
  cutoff.looks = NULL,
  mc.alpha = NULL
)
```

Arguments

<code>data</code>	A data frame from <code>simdata_fast</code> for a two-group trial, containing at least <code>sim</code> , <code>group</code> , <code>accrual_time</code> , <code>tte</code> , and <code>event</code> .
<code>control</code>	A scalar value indicating which level of group represents the control group.
<code>event.looks</code>	A vector of positive whole numbers, the target cumulative event counts, one per look. Mutually exclusive with <code>time.looks</code> and <code>cutoff.looks</code> .
<code>time.looks</code>	A numeric vector of calendar times, one per look. Mutually exclusive with <code>event.looks</code> and <code>cutoff.looks</code> .
<code>stat</code>	A character vector naming the statistics to compute. Any subset of "logrank", "coxph", "rmst", "km", "maxcombo", "ahsw", "milestone", "rmw", "ahr",

	"medsurv", "wkm", and "wmst". Defaults to "logrank".
tau	A single positive numeric value, the restriction horizon for "rmst", the truncation time for "ahsw" and "ahr", and the milestone timepoint for "milestone". Required only when "rmst", "ahsw", "milestone", or "ahr" is requested.
t.eval	A single positive numeric value, the landmark time for "km". Required only when "km" is requested.
conf.level	A single numeric value in (0, 1), the confidence level for "coxph", "rmst", and "ahsw". Defaults to 0.95.
side	An integer, either 1 or 2. When side = 2 (default), two-sided p-values $2 \text{pnorm}(- z)$ are reported for log-rank, Cox, and RMST, and the two-sided max-combo test is used. When side = 1, the one-sided p-value in the direction of treatment benefit is reported for each of those statistics, and the one-sided max-combo test is used. The test statistics themselves are always reported with their natural sign, so the choice of side affects only the p-value columns. For log-rank and Cox the benefit direction is a negative Z (the one-sided p-value is the lower tail $\text{pnorm}(z)$); for RMST it is a positive Z (the upper tail $\text{pnorm}(-z)$). The other statistics follow the benefit directions given in Details. For group-sequential boundary comparisons (for example with gsDesign or rpact), align the sign of the reported Z with the boundary convention before comparing.
by.subgroup	A logical value. When TRUE, the analysis is also reported within each subgroup level, and the output gains a population column (long form). When FALSE (default), only the whole-population analysis is returned and no population column is added. Requires at least one subgroup column in data.
weight	A character string naming the weight scheme for the "logrank" statistic. "logrank" (default) is the ordinary unweighted test. "fh" is the Fleming-Harrington G(rho, gamma) test, "mwlrt" is the modestly-weighted log-rank test (requires t_star), "gehan" is Gehan-Breslow, and "tarone-ware" is Tarone-Ware. See survdiff_fast .
rho	A numeric Fleming-Harrington first parameter, used only when weight = "fh". Defaults to 0.
gamma	A numeric Fleming-Harrington second parameter, used only when weight = "fh". Defaults to 0.
t_star	A single non-negative numeric value, the timepoint of the modestly-weighted log-rank test. Required only when weight = "mwlrt".
strata	An optional character vector naming one or more subgroup columns of data to use as the stratification variable for the "logrank" and "coxph" statistics. NULL (default) gives the unstratified analyses. When several columns are named their interaction defines the strata. The columns must not contain missing values. Stratification applies only to the "logrank" and "coxph" statistics; the other statistics ignore it.
ms.method	A character string naming the inference method for the "milestone" statistic, one of "wald" (default), "loglog", or "mover". See milestone_fast .
s_star	A single numeric value in (0, 1], the survival-probability threshold of the modestly-weighted component of the "rmw" statistic. The weight is capped at $1 / s_star$. Defaults to 0.5. See rmw_fast .

<code>mc.rho</code>	A numeric vector of Fleming-Harrington first parameters for the "maxcombo" statistic, one per component weight. Defaults to <code>c(0, 0, 1, 1)</code> .
<code>mc.gamma</code>	A numeric vector of Fleming-Harrington second parameters for the "maxcombo" statistic, aligned with <code>mc.rho</code> . Defaults to <code>c(0, 1, 0, 1)</code> .
<code>abseps</code>	A single positive numeric value, the absolute error tolerance passed to the multivariate normal integration of the "maxcombo" p-value. Defaults to <code>1e-5</code> .
<code>maxpts</code>	A single positive integer, the maximum number of function evaluations for the quasi-Monte-Carlo integration used by the "maxcombo" p-value with four or more weights or with a two-sided test (see Details). Defaults to <code>25000</code> .
<code>medsurv.method</code>	A character string naming the variance method for the "medsurv" statistic, one of "km" (default) or "nph". See medsurv_fast .
<code>medsurv.bw</code>	An optional single positive numeric value, the kernel bandwidth for the median hazard used by <code>medsurv.method = "km"</code> , applied to both groups. NULL (default) uses a Silverman type default computed per group from the cut data, matching medsurv_fast .
<code>wkm.weight</code>	A character string naming the weight for the "wkm" statistic, one of "PF" (default, Pepe-Fleming combined censoring weight), "sqrtPF", or "constant". See wkm_fast .
<code>wmst.tau1</code>	A single non-negative numeric value, the lower window limit for the "wmst" statistic. Defaults to <code>0</code> .
<code>wmst.tau2</code>	A single positive numeric value, the upper window limit for the "wmst" statistic, which must exceed <code>wmst.tau1</code> . NULL (default) falls back to <code>tau</code> . Required (through either argument) only when "wmst" is requested.
<code>cutoff.looks</code>	A numeric matrix of per-simulation calendar cutoffs with one row per simulated trial and one column per look, such as the output of cutoff_fast . When the matrix has row names they are matched to the values of <code>data\$sim</code> ; otherwise the rows are taken in the order of the sorted distinct values of <code>data\$sim</code> . A numeric vector is treated as a single look. NA or Inf marks a look that is not reached. The <code>look.value</code> column of the output is taken from the "look.value" attribute of the matrix when it is present, and is NA otherwise. Mutually exclusive with <code>event.looks</code> and <code>time.looks</code> .
<code>mc.alpha</code>	An optional numeric vector of nominal levels in $(0, 1)$, one per look or a single value for all looks, at which the "maxcombo" p-value is compared. When supplied, the p-value is computed by integration only when the Bonferroni bounds do not decide the comparison (see Details), and the output gains a logical column <code>maxcombo.p.exact</code> . NULL (default) computes every p-value.

Details

The input data is the data frame returned by [simdata_fast](#) for a two-group trial. The columns `sim`, `group`, `accrual_time`, `tte`, and `event` are required, must not contain missing values, and `group` must have exactly two distinct values, one of which is `control`. For a multi-arm trial, subset the data to two arms first or use [pairwise_fast](#).

For a look at calendar time cutoff, each subject with accrual time `a` contributes only if enrolled by then (`a <= cutoff`). The observed time at the look is `min(tte, cutoff - a)` and the observed event

indicator is the original event when the natural event or dropout occurred on or before cutoff ($a + tte \leq \text{cutoff}$), and zero otherwise (administrative censoring at the look).

When `event.looks` is supplied, the calendar cutoff for a target of d events is the calendar time of the d -th event in that simulated trial, counted over the whole trial population. If a simulation contains fewer than d events, the target is never reached: the full data are used, so the statistics are those of the final data, `reached` is FALSE, and `cutoff` is NA. When `time.looks` is supplied, the cutoff is the specified calendar time and `reached` is always TRUE. In both cases the cutoff is determined once on the whole population and then used for the overall analysis and for every subgroup analysis at that look.

When `cutoff.looks` is supplied, the calendar cutoff of each look is given separately for every simulated trial, typically by `cutoff_fast`. This analyzes an endpoint at cutoffs determined by another endpoint (for example overall survival at the event-driven looks of progression-free survival), by a subset of the subjects, or by a combination of event and calendar-time rules. A missing or infinite entry marks a look whose trigger was not met in that simulation: as for an unreached `event.looks` target, the full data are used, `reached` is FALSE, and `cutoff` is NA. In contrast, `pairwise_fast` reports NA counts and statistics for an unreached look, because its contrasts share one cutoff and have no common final data.

Exactly one of `event.looks`, `time.looks`, and `cutoff.looks` must be supplied.

The statistics are selected with `stat`, which may name one or more of "logrank", "coxph", "rmst", "km", "maxcombo", "ahsw", "milestone", "rmw", "ahr", "medsurv", "wkm", and "wmst".

The "logrank" statistic is configurable. By default it is the ordinary unweighted, unstratified two-group log-rank test and reproduces the behavior of earlier versions of this function exactly. A non-default weight selects a weighted log-rank test (Fleming-Harrington, modestly-weighted, Gehan-Breslow, or Tarone-Ware) for non-proportional hazards, and a non-NULL `strata` selects the stratified test, summing the per-stratum contributions. The two options combine to give the stratified weighted log-rank test. Whatever configuration is chosen, the result is written to the same `logrank.z`, `logrank.chisq`, and `logrank.p` columns; the unweighted unstratified case is the $K = 1$ single-stratum degenerate form of the general statistic. The columns `rho`, `gamma`, and `t_star` parametrize weight, and `strata` names one or more subgroup columns of data used as the stratification variable. The stratification is determined on the whole cut data, independently of the population marginalization, so a stratified overall analysis is the canonical primary test; in a single-subgroup population the stratum is constant and the stratified test degenerates to the ordinary one within that subset. The same `strata` also stratify the "coxph" statistic, which is then the stratified Pike-Halley estimate of `coxph_fast` (a common hazard ratio with a separate baseline hazard in each stratum).

The "maxcombo" statistic is the max-combo test of `mc.rho` and `mc.gamma` Fleming-Harrington weights. Its `maxcombo.stat` is the most extreme component (min of the component Z-scores when `side = 1`, so that a negative value favors treatment, and the maximum absolute component when `side = 2`), and `maxcombo.p` is the joint multivariate-normal p-value, which already follows `side`. As in `maxcombo_fast`, the p-value of a one-sided test with two or three weights is computed by the deterministic TVPACK algorithm, and otherwise by the randomized quasi-Monte-Carlo GenzBretz algorithm of `mvtnorm`, which uses R's random-number generator; call `set.seed()` beforehand for p-values that are reproducible to the last digit.

The multivariate normal integration dominates the computing time of the "maxcombo" statistic in a large simulation. When only the decision at a nominal level matters, `mc.alpha` avoids most of these integrals. With K weights and `p_min` the smallest component p-value (`pnorm(m)` for `side =`

1 and $2 * pnorm(-m)$ for $side = 2$, where m is `maxcombo.stat`), the max-combo p-value satisfies $p_{min} \leq p \leq \min(1, K * p_{min})$ (the Bonferroni inequality). When $p_{min} > mc.alpha$, p_{min} is reported; when $K * p_{min} \leq mc.alpha$, $K * p_{min}$ is reported; otherwise the p-value is computed by integration. The reported value is therefore on the same side of `mc.alpha` as the exact p-value, so the rejection decision `maxcombo.p <= mc.alpha` (as in `simsummary_fast` with `alpha = mc.alpha`) is the same as with exact p-values, but the reported value is a bound unless `maxcombo.p.exact` is TRUE. Use exact p-values (the default `mc.alpha = NULL`) when the p-values themselves, or decisions at other levels such as Bonferroni-adjusted levels, are needed.

The "ahsw" statistic is the average hazard with survival weight of Uno and Horiguchi on the window from 0 to τ . It reports the per-group average hazards, the ratio (RAH) and difference (DAH) contrasts with their confidence intervals, and p-values for both contrasts that follow `side`, matching `ahsw_fast`. The benefit direction is a ratio below 1 (a negative log ratio) and a negative difference, so the one-sided p-values are lower-tail probabilities.

The "milestone" statistic compares the Kaplan-Meier survival probabilities of the two groups at the milestone timepoint τ . It reports the per-group survival, the difference (treatment minus control) with its confidence interval, the test statistic, and the p-value. The inference method is selected with `ms.method` ("wald", "loglog", or "mover"), matching `milestone_fast`. The benefit direction is a positive difference (higher treatment survival). For "wald" and "mover" a positive Z favors treatment; for "loglog" the statistic is the difference of the complementary log-log transforms, so a negative Z favors treatment, and the one-sided p-value is the lower tail as in `milestone_fast`. When the Kaplan-Meier estimate of a group is 0 or 1 at τ , its one-sample interval degenerates to the estimate, so the interval of the difference is still reported, while the "loglog" statistic is NA.

The "rmw" statistic is the robust modestly-weighted log-rank test of Magirr and Ohn, the maximum of the standard log-rank component and a single modestly-weighted component with survival-threshold `s_star`. Its `rmw.stat` is the most extreme standardized component (the minimum when `side = 1`, so a negative value favors treatment, and the maximum absolute component when `side = 2`), and `rmw.p` is the joint bivariate-normal p-value, which already follows `side`, matching `rmw_fast`.

The "ahr" statistic is the Kalbfleisch-Prentice average hazard ratio over the window from 0 to τ . It reports the average hazard ratio (treatment relative to control), the two group shares of the total hazard, the test statistic on the group-share scale, and the p-value. The benefit direction is an average hazard ratio below 1 (a negative Z), as in `ahr_fast`.

The "medsurv" statistic compares the non-parametric Kaplan-Meier median survival times of the two groups. It reports the per-group medians, their difference (treatment minus control) with a plain-scale confidence interval, the Wald statistic, and the p-value. The variance method is selected with `medsurv.method` ("km" or "nph"), matching `medsurv_fast`; `medsurv.bw` optionally overrides the kernel bandwidth used by `medsurv.method = "km"`. The benefit direction is a positive difference (a longer median under treatment), so a positive Z favors treatment.

The "wkm" statistic is the weighted Kaplan-Meier (Pepe-Fleming) test. It reports the weighted integrated survival difference (treatment minus control) with its confidence interval, the Wald statistic, and the p-value. The weight is selected with `wkm.weight` ("PF", "sqrtPF", or "constant"), matching `wkm_fast`. The benefit direction is a positive weighted difference (a positive Z favors treatment).

The "wmst" statistic compares the window mean survival time, the area under the Kaplan-Meier curve between `wmst.tau1` and `wmst.tau2`. It reports the per-group WMST values, their difference (treatment minus control) with its confidence interval, the Wald statistic, and the p-value, matching `wmst_fast`. With `wmst.tau1 = 0` the window mean survival time is the restricted mean survival time at `wmst.tau2`. The benefit direction is a positive difference (a positive Z favors treatment).

When `by.subgroup = TRUE`, the output is given in long form with a population column. Each `(sim, look)` produces one row for the whole trial (population = "overall") plus one row per subgroup level of each subgroup factor in data. Subgroup factors are the columns named `subgroup` or `subgroup1`, `subgroup2`, and so on. Each factor is marginalized separately, so a factor with levels 1 and 2 yields the populations `subgroup_1` and `subgroup_2` (or `subgroup1_1`, `subgroup1_2`, and so on for numbered factors). The look cutoff is always determined on the whole population, so subgroup rows at a given look share the same cutoff, reached, and `look.value`; their `n.enrolled` and `n.event` are the counts within that subgroup. When `by.subgroup = FALSE` (default) the output has no population column and one row per `(sim, look)`, matching the whole-population analysis.

Value

A data frame. When `by.subgroup = FALSE`, it has `nsim * length(looks)` rows. When `by.subgroup = TRUE`, it has `nsim * length(looks) * (1 + total subgroup levels)` rows and an extra population column placed after `look.value`. The common columns are `sim`, `look` (1-based look index), `look.value` (the requested event count or calendar time, or the "look.value" attribute of `cutoff.looks`), optionally `population`, `cutoff` (the calendar time used, NA when the look was not reached), `reached`, `n.enrolled`, `n.event`, `n.dropout` (the number of enrolled subjects whose dropout occurred on or before the cutoff) and `n.pipeline` (`n.enrolled - n.event - n.dropout`, the subjects still in follow-up at the cutoff), followed by the columns of the requested statistics. A statistic that cannot be computed for a row (no events, or an empty group) is NA. The statistic columns are `logrank.z`, `logrank.chisq`, and `logrank.p` for "logrank"; `cox.coef`, `cox.hr`, `cox.se`, `cox.z`, `cox.p`, `cox.lower`, and `cox.upper` for "coxph"; `rmst.ctrl`, `rmst.trt`, `rmst.diff`, `rmst.diff.lower`, `rmst.diff.upper`, `rmst.z`, and `rmst.p` for "rmst"; `km.surv.ctrl` and `km.surv.trt` for "km"; `maxcombo.stat` and `maxcombo.p` (and `maxcombo.p.exact` when `mc.alpha` is supplied) for "maxcombo"; and `ahsw.ah.ctrl`, `ahsw.ah.trt`, `ahsw.rah`, `ahsw.rah.lower`, `ahsw.rah.upper`, `ahsw.p.rah`, `ahsw.dah`, `ahsw.dah.lower`, `ahsw.dah.upper`, and `ahsw.p.dah` for "ahsw"; `milestone.surv.ctrl`, `milestone.surv.trt`, `milestone.diff`, `milestone.diff.lower`, `milestone.diff.upper`, `milestone.z`, and `milestone.p` for "milestone"; `rmw.stat` and `rmw.p` for "rmw"; and `ahr.ahr`, `ahr.theta.ctrl`, `ahr.theta.trt`, `ahr.z`, and `ahr.p` for "ahr"; `medsurv.ctrl`, `medsurv.trt`, `medsurv.diff`, `medsurv.diff.lower`, `medsurv.diff.upper`, `medsurv.z`, and `medsurv.p` for "medsurv"; `wkm.wdiff`, `wkm.lower`, `wkm.upper`, `wkm.z`, and `wkm.p` for "wkm"; and `wmst.ctrl`, `wmst.trt`, `wmst.diff`, `wmst.diff.lower`, `wmst.diff.upper`, `wmst.z`, and `wmst.p` for "wmst". The Z columns `logrank.z`, `cox.z`, and `rmst.z` carry the natural sign of each test, and the p-value columns follow side.

See Also

[simdata_fast](#), [survdiff_fast](#), [coxph_fast](#), [rmst_fast](#), [survfit_fast](#), [maxcombo_fast](#), [ahsw_fast](#), [milestone_fast](#), [rmw_fast](#), [ahr_fast](#), [medsurv_fast](#), [wkm_fast](#), [wmst_fast](#), [cutoff_fast](#), [pairwise_fast](#), [switch_fast](#).

Examples

```
df <- simdata_fast(
  nsim      = 50,
  n         = c(150, 150),
  a.time    = c(0, 12),
  a.rate    = 300 / 12,
  e.hazard  = list(list(0.10, 0.07), 0.05),
  prevalence = c(0.5, 0.5),
```

```

    seed      = 1
  )

# Whole-population analysis at two event-based looks
res1 <- analysis_fast(df, control = 1, event.looks = c(80, 140))
head(res1)

# Stratified log-rank and stratified Cox on the subgroup factor
res2 <- analysis_fast(df, control = 1, time.looks = 24,
                      stat = c("logrank", "coxph"), strata = "subgroup")
head(res2)

# Fleming-Harrington G(0, 1) weighted log-rank for delayed effects
res3 <- analysis_fast(df, control = 1, time.looks = 24,
                      stat = "logrank", weight = "fh", rho = 0, gamma = 1)
head(res3)

# Max-combo and AHSW together
res4 <- analysis_fast(df, control = 1, time.looks = 24,
                      stat = c("maxcombo", "ahsw"), tau = 18)
head(res4)

# Milestone survival, robust modestly-weighted, and average hazard ratio
res5 <- analysis_fast(df, control = 1, time.looks = 24,
                      stat = c("milestone", "rmw", "ahr"), tau = 18,
                      ms.method = "loglog", s_star = 0.5, side = 1)
head(res5)

```

coxph_fast

Fast Closed-Form Hazard Ratio Estimation via the Pike-Halley Estimator

Description

Estimates the hazard ratio for a two-group parallel trial using the Pike-Halley Estimator (Homma, 2025), a pure closed-form approximation to the Cox partial likelihood maximizer. The function returns the point estimate, its standard error on the log scale, and a Wald-type confidence interval, using output names consistent with `summary(survival::coxph(...))`. The C++ backend accepts pooled sorted vectors directly, performing group splitting and all accumulation in a single C++ pass without intermediate R-level vector copies.

Usage

```

coxph_fast(
  time,
  event,
  group,
  control,

```

```

    side = 2,
    conf.level = 0.95,
    presorted = FALSE,
    strata = NULL
  )

```

Arguments

time	A numeric vector of follow-up times for all subjects (pooled over both groups).
event	An integer or numeric vector of event indicators (1 = event, 0 = censored), aligned with time.
group	A vector of group labels aligned with time. Any type that supports equality comparison is accepted.
control	A scalar value indicating which level of group represents the control group. Subjects with group != control are treated as the treatment group.
side	1 for a one-sided test in the direction of treatment benefit (hazard ratio below 1, i.e. a negative coefficient) or 2 for a two-sided test (default 2). The returned vector has no p-value: side is stored as an attribute, and the print() method reports the p-value that follows it. The confidence interval is always two-sided at conf.level.
conf.level	A single numeric value in (0, 1) specifying the confidence level for the Wald interval. Defaults to 0.95.
presorted	A logical value. If TRUE, time, event, and group are assumed to be already sorted in ascending order of time (with strata, sorted by stratum and by time within stratum, so that the rows of each stratum are contiguous), and the internal order() call is skipped. If FALSE (default), sorting is handled internally. The order is checked, and an error is given when it does not hold.
strata	An optional vector of stratum labels aligned with time, without missing values. When supplied, the stratified estimator described in Details is computed. Several stratification factors can be combined with interaction(). NULL (default) gives the unstratified estimator.

Details

Let t_k ($k = 1, \dots, K$) denote the distinct observed event times in the pooled sample. At each t_k , let n_{T_k} and n_{C_k} be the numbers at risk in the treatment and control groups just before t_k , and let O_{T_k} and O_{C_k} be the numbers of events in each group, with $n_k = n_{T_k} + n_{C_k}$ and $O_k = O_{T_k} + O_{C_k}$. Define $E_T = \sum n_{T_k} O_k / n_k$ and $E_C = \sum n_{C_k} O_k / n_k$ as the log-rank expected event totals.

The Pike-Halley Estimator is obtained in three steps. First, the Pike anchor is computed as $\theta_0 = (O_T E_C) / (O_C E_T)$. Second, the score U_0 , the observed information I_0 , and the third-order curvature term J_0 of the Breslow partial likelihood are evaluated at $\eta_0 = \log(\theta_0)$:

$$p_k = n_{T_k} \theta_0 / (n_{C_k} + n_{T_k} \theta_0) \quad U_0 = \sum (O_{T_k} - O_k p_k) \quad I_0 = \sum O_k p_k (1 - p_k) \quad J_0 = \sum O_k p_k (1 - p_k) (1 - 2 p_k)$$

Third, the closed-form Halley correction is applied:

$$\hat{\Delta} = U_0 / I_0 - J_0 U_0^2 / (2 I_0^3) \quad \hat{\theta} = \theta_0 \exp(\hat{\Delta})$$

The Halley step converges cubically, so the residual error $|\log(\theta_{\text{hat}}) - \log(\theta_{\text{Cox}})|$ is of the order of the cube of the error of the Pike anchor. Near the null hypothesis the anchor is already close to the Cox estimate and the residual error is negligible. At a fixed hazard ratio away from 1 the Pike anchor has a bias that does not vanish as the sample size grows (Berry, Kitchin, and Mock, 1991), so the residual error levels off at a small value that depends on the hazard ratio and the censoring pattern instead of decreasing with the sample size.

The Wald standard error on the log scale is $SE = 1 / \sqrt{I_0}$, where I_0 is the observed information evaluated at the Pike anchor. The Wald confidence interval reported by `summary(coxph(...))` uses the observed information at the maximum likelihood estimate instead. The two differ by an amount of the order of the error of the Pike anchor, which is small when the hazard ratio is not far from 1; Berry, Kitchin, and Mock (1991) found the bias of the Pike estimator minimal for hazard ratios below 3.

With `strata`, the estimator targets the stratified Cox model, in which each stratum has its own baseline hazard and the hazard ratio is common to all strata. The at-risk sets are formed within each stratum, the observed and expected totals O_T , O_C , E_T , and E_C are summed over strata to give the stratified Pike anchor, and U_0 , I_0 , and J_0 are those of the stratified Breslow partial likelihood (sums of the per-stratum terms). The Halley correction and the Wald interval are then applied unchanged, so the result approximates `coxph(Surv(time, event) ~ group + strata(s), ties = "breslow")`. With a fixed number of strata the approximation error shrinks at the same rate as in the unstratified case; it is larger when the strata are small and the hazard ratio is far from 1, because the stratified anchor is then further from the maximum likelihood estimate.

The C++ core (`pihe_core`) accepts the pooled sorted data together with an integer group indicator and performs group splitting, at-risk counting, and per-distinct-event-time accumulation in a single left-to-right scan. This eliminates the `rev(cumsum(rev(...)))`, `tapply()`, `which()`, `diff()`, and group-split vector copies present in the pure-R version.

The returned object has class `"coxph_fast"` and is a named numeric vector of length 5. A `print()` method formats the result similarly to `summary(coxph(...))`.

Value

An object of class `"coxph_fast"`, which is a named numeric vector of length 5 with elements matching the column names of `summary(coxph(...))$coefficients` and `summary(coxph(...))$conf.int`:

`coef` Log hazard ratio $\log(\theta_{\text{hat}})$.

`exp(coef)` Hazard ratio θ_{hat} (point estimate).

`se(coef)` Standard error of `coef` on the log scale, equal to $1 / \sqrt{I_0}$.

`lower .95` Lower bound of the Wald confidence interval for the hazard ratio. The label reflects `conf.level` (e.g., `"lower .90"` when `conf.level = 0.90`).

`upper .95` Upper bound of the Wald confidence interval.

With `strata`, the number of strata is stored in the attribute `strata`. Returns a vector of `NA_real_` values (still with class `"coxph_fast"`) when the estimate cannot be computed (e.g., no events, all events in one group, or $I_0 = 0$).

References

Cox, D. R. (1972). Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2), 187-220.

Berry, G., Kitchin, R. M., & Mock, P. A. (1991). A comparison of two simple hazard ratio estimators based on the logrank test. *Statistics in Medicine*, 10(5), 749-755.

Homma, G. (2025). One step from Pike to Cox: a closed-form hazard ratio estimator. Manuscript under review.

See Also

`coxph` for the standard iterative Cox estimator. `print.coxph_fast` for the print method.

Examples

```
library(survival)

# Compare coxph_fast with coxph on the ovarian dataset.
# coxph() treats rx as numeric with rx=1 as the reference (control),
# so set control = 1 for a consistent comparison.
fit_fast <- coxph_fast(ovarian$futime, ovarian$fustat, ovarian$rx, control = 1)
fit_fast

fit_cox <- summary(coxph(Surv(futime, fustat) ~ rx, data = ovarian))
cat("coxph_fast HR :", fit_fast["exp(coef)"], "\n")
cat("coxph      HR :", fit_cox$coefficients[, "exp(coef)"], "\n")

# Stratified by residual disease, compared with coxph(... + strata())
coxph_fast(ovarian$futime, ovarian$fustat, ovarian$rx, control = 1,
           strata = ovarian$resid.ds)
coef(coxph(Surv(futime, fustat) ~ rx + strata(resid.ds), data = ovarian,
           ties = "breslow"))

# presorted = TRUE: sort once outside, reuse inside a loop
ord <- order(ovarian$futime)
coxph_fast(ovarian$futime[ord], ovarian$fustat[ord], ovarian$rx[ord],
           control = 1, presorted = TRUE)

# Speed comparison against coxph()
if (requireNamespace("microbenchmark", quietly = TRUE)) {
  microbenchmark::microbenchmark(
    coxph_fast = coxph_fast(ovarian$futime, ovarian$fustat, ovarian$rx, 2),
    coxph      = coxph(Surv(futime, fustat) ~ rx, data = ovarian),
    times = 1000
  )
}
```

Description

Computes the calendar time of each analysis look in every simulated trial from a combination of trigger rules: a target number of events, a planned calendar time, a maximum calendar time, a minimum time after the previous look, and a minimum follow-up after a given number of subjects are enrolled. The cutoffs are computed for all simulated trials in a single C++ pass and are returned as a matrix that can be passed to the `cutoff.looks` argument of [analysis_fast](#) and [pairwise_fast](#), or to the `cutoff` argument of [switch_fast](#). Separating the determination of the analysis time from the analysis itself lets the same cutoffs drive several endpoints (for example overall survival analyzed at the progression-free survival event-driven looks), several contrasts, or a modification of the data after an interim analysis.

Usage

```
cutoff_fast(
  data,
  event.looks = NULL,
  time.looks = NULL,
  max.time = NULL,
  min.gap = NULL,
  min.enrolled = NULL,
  min.followup = NULL,
  event.subset = NULL,
  tte.col = "tte",
  event.col = "event"
)
```

Arguments

<code>data</code>	A data frame of simulated trials, such as the output of simdata_fast , with columns <code>sim</code> , <code>accrual_time</code> , and the columns named by <code>tte.col</code> and <code>event.col</code> .
<code>event.looks</code>	Target cumulative event counts. Either a vector with one positive whole number per look (NA to omit the event condition at a look), or a matrix of positive whole numbers with one row per simulated trial (in the order of the sorted distinct values of <code>data\$sim</code>) and one column per look.
<code>time.looks</code>	Planned calendar times, one per look (NA to omit).
<code>max.time</code>	Maximum calendar times, one per look (NA to omit). The cutoff never exceeds this value.
<code>min.gap</code>	Minimum time after the previous look, one per look (NA to omit). At the first look it is measured from time zero.
<code>min.enrolled</code>	Number of enrolled subjects after which the minimum follow-up <code>min.followup</code> must elapse, one per look (NA to omit).
<code>min.followup</code>	Minimum follow-up after the <code>min.enrolled</code> -th subject is enrolled, one per look. Defaults to zero when <code>min.enrolled</code> is supplied, and requires <code>min.enrolled</code> .
<code>event.subset</code>	An optional logical vector with one element per row of data. When supplied, only the events of the rows with TRUE are counted for <code>event.looks</code> .

<code>tte.col</code>	A single character string naming the observed-time column used to count events. Defaults to "tte".
<code>event.col</code>	A single character string naming the event-indicator column (0 or 1) used to count events. Defaults to "event".

Details

For look `l` of a simulated trial the candidate calendar times are

- `t_cal`, the planned calendar time `time.looks[l]`;
- `t_event`, the calendar time (accrual time plus observed time) of the `d`-th counted event, where `d` is `event.looks[l]` (or the entry of the `event.looks` matrix for that simulation and look);
- `t_gap`, the cutoff of the previous look plus `min.gap[l]` (zero plus `min.gap[l]` at the first look);
- `t_enr`, the accrual time of the `min.enrolled[l]`-th enrolled subject plus `min.followup[l]`;

and the cutoff is `min(max(t_cal, t_event, t_gap, t_enr), max.time[l])`, where a condition that is not supplied (or is NA at that look) is left out. This is the rule of `get_analysis_date()` in the `simtrial` package (with `max.time` playing the role of its `max_extension_for_target_event`), applied to every simulated trial at once. For example, `event.looks = 300` with `max.time = 48` is "300 events or month 48, whichever comes first", and `event.looks = 300` with `time.looks = 36` is "300 events but not before month 36".

An event or enrollment target that is not met in the simulated data has no finite calendar time. Unless a `max.time` caps the look, its cutoff is then NA, which `analysis_fast` reports as `reached = FALSE` with the statistics of the full data, as it does for an unreached `event.looks` target. In this respect the function differs from `simtrial`, which uses the last observed event time for an unmet target.

Events are counted on the columns named by `tte.col` and `event.col`, so a look can be triggered by an endpoint other than the one analyzed (for example the `e1_tte` and `e1_event` columns of an illness-death simulation). By default every subject's events are counted; `event.subset` restricts the count to a subset of rows, such as the control group (`data$group == 1`), one subgroup, or the two arms of the primary contrast in a multi-arm trial. When the subjects tie at the target event time, all events at that time are included in the analysis, so the analyzed event count can exceed the target, as in `analysis_fast`.

A matrix `event.looks` with one row per simulated trial gives simulation-specific event targets, for example targets re-estimated at an interim analysis.

Each look is determined by its own conditions; only `min.gap` refers to the previous look. A look with `max.time` as its only condition is placed at that time. When the previous look is not reached, `t_gap` is infinite, so a look with `min.gap` is not reached either unless its `max.time` caps it. The looks are not reordered: when the conditions of a later look give an earlier cutoff than the previous look (for example `event.looks = c(150, 220)` with `max.time = c(NA, 30)`), a warning is given. Supplying `min.gap` (for example 0) at the later looks keeps them in order unless their `max.time` is earlier.

Value

A numeric matrix with one row per simulated trial and one column per look, holding the calendar cutoffs (NA for a look that is not reached). The row names are the simulation identifiers, and

the attribute "look.value" holds, for each look, the event target when a common target is supplied, otherwise the planned calendar time, otherwise NA. [analysis_fast](#) reports this attribute in its look.value column.

See Also

[analysis_fast](#), [pairwise_fast](#), [switch_fast](#), [simdata_fast](#).

Examples

```
df <- simdata_fast(nsim = 20, n = c(150, 150), a.time = c(0, 12),
                  a.rate = 300 / 12, e.median = list(12, 18),
                  d.hazard = 0.01, seed = 1)

# 150 events or month 30, whichever comes first, then a final look at
# 220 events but at least 6 months after the interim
cut <- cutoff_fast(df, event.looks = c(150, 220), max.time = c(30, NA),
                  min.gap = c(NA, 6))

head(cut)

res <- analysis_fast(df, control = 1, cutoff.looks = cut)
head(res)

# Looks triggered by the events of the control group only
cut_c <- cutoff_fast(df, event.looks = 100, event.subset = df$group == 1)
head(cut_c)
```

gen_scenario_fast

Build a Set of Two-Group Trial Scenarios for Design Exploration

Description

Assembles one or more two-group time-to-event scenarios into a single `scenario_fast` object for design-stage exploration. Each scenario is a complete set of [simdata_fast](#) arguments, so the same object can be drawn with [plot.scenario_fast](#), summarized with [print.scenario_fast](#), and passed to [simdata_fast](#) to generate data. This is a design-helper that only collects and merges arguments; it performs no simulation and no compiled computation.

Usage

```
gen_scenario_fast(scenarios, shared = list(), labels = NULL)
```

Arguments

scenarios	A non-empty list. Each element is a named list of simdata_fast arguments (optionally with label and null) that defines one scenario.
shared	A named list of simdata_fast arguments common to every scenario. Scenario-specific arguments take precedence.
labels	Optional character vector of scenario labels, one per scenario.

Details

Each element of `scenarios` is a named list of arguments that defines one scenario. Those arguments override the shared arguments in `shared`, so that parameters held constant across scenarios (sample size, accrual, dropout) are written once in `shared` and only the varying parameters are written per scenario. The survival specification follows `simdata_fast` exactly: `e.hazard` or `e.median` given as a two-element list (control first, treatment second) for the two groups, with `e.time` supplying the breakpoints of a piecewise-exponential hazard (last element `Inf`). A single hazard or median gives an exponential group.

Two optional fields may appear inside a scenario list and are interpreted by this function rather than passed on: `label` sets the scenario label, and `null` (logical) flags a null scenario for later filtering. Every other field is treated as a `simdata_fast` argument. The scenario label is taken from `labels`, then from the `label` field, then from the names of `scenarios`, then from a default "Scenario k".

A factorial set of scenarios is built by constructing the `scenarios` list with the base tools, for example mapping over the rows of an `expand.grid` of the parameters that vary.

Value

An object of class `scenario_fast`: a list with element `scenarios` (a named list, one entry per scenario, each holding the merged argument list `args`, the `label`, and the `null` flag) and element `shared` (the shared arguments).

See Also

[simdata_fast](#), [plot.scenario_fast](#), [print.scenario_fast](#)

Examples

```
# Three two-group scenarios sharing accrual and sample size: a proportional-
# hazards case, a delayed-effect case, and a crossing-hazards case.
scn <- gen_scenario_fast(
  scenarios = list(
    "Proportional" = list(e.median = list(12, 18)),
    "Delayed effect" = list(
      e.hazard = list(log(2) / 12, c(log(2) / 12, log(2) / 22)),
      e.time = c(0, 6, Inf)
    ),
    "Crossing" = list(
      e.hazard = list(log(2) / 12, c(log(2) / 7, log(2) / 24)),
      e.time = c(0, 5, Inf)
    )
  ),
  shared = list(n = c(150, 150), a.time = c(0, 12), a.rate = 300 / 12)
)

print(scn)
plot(scn)

# Generate data for every scenario with a short composition step. From here
# analysis_fast() and simsummary_fast() are applied per scenario as usual.
sim_list <- lapply(scn$scenarios, function(s) {
```

```
do.call(simdata_fast, c(s$args, list(nsim = 100, seed = 1)))
})
nrow(sim_list[[1]])
```

kmcurve_fast

Build Kaplan-Meier Curves for One Trial Realization

Description

Builds the Kaplan-Meier step functions for a single realized data set (for example one replicate from [simdata_fast](#)) and returns a `kmcurve_fast` object that can be drawn with [plot.kmcurve_fast](#) and summarized with [print.kmcurve_fast](#). Both the two-group case and the single-group case are supported. This is an analysis-stage and plotting helper, written in plain R for use on a single data set, and is not intended to be called inside a simulation loop. The fast point estimator [survfit_fast](#) remains the tool for repeated evaluation inside loops.

Usage

```
kmcurve_fast(time, event, group = NULL, control = NULL)
```

Arguments

<code>time</code>	A numeric vector of follow-up times for all subjects.
<code>event</code>	An integer or numeric vector of event indicators (1 = event, 0 = censored), aligned with <code>time</code> .
<code>group</code>	A vector identifying the groups, aligned with <code>time</code> , with one or two distinct values. If <code>NULL</code> (default), all subjects are treated as a single group.
<code>control</code>	The value of <code>group</code> that denotes the reference (control) group; the other value is treated as the treatment group. Required in the two-group case and ignored in the single-group case.

Details

For each group the object stores the distinct event times, the number of events and the number at risk at those times, the Kaplan-Meier survival estimate, the Greenwood standard error, the Nelson-Aalen increments d_i / Y_i (used by the plot method to estimate a smoothed time-varying hazard ratio), and the censoring times. The pooled data are also retained so that the plot and print methods can reuse [coxph_fast](#), [rmst_fast](#), and [ahr_fast](#) for the constant Cox hazard ratio, the restricted mean survival time, and the average hazard ratio. When `group` is `NULL` or has a single distinct value, a single-group object is built and the plot method draws one survival curve with no hazard ratio.

Value

An object of class `kmcurve_fast`: a list with a logical `two_group` flag, the group values and their labels, the per-group Kaplan-Meier summaries in element `km`, the pooled data, and the per-group sample sizes `n` and event counts `events`.

See Also

[plot.kmcurve_fast](#), [print.kmcurve_fast](#), [survfit_fast](#), [coxph_fast](#), [rmst_fast](#), [ahr_fast](#)

Examples

```
set.seed(1)
n <- 150
t0 <- rexp(n, log(2) / 12)
t1 <- rexp(n, log(2) / 18)
cens <- runif(2 * n, 0, 30)
time <- pmin(c(t0, t1), cens)
event <- as.integer(c(t0, t1) <= cens)
group <- rep(c(0, 1), each = n)

fit <- kmcurve_fast(time, event, group, control = 0)
print(fit)
plot(fit)
plot(fit, hr = TRUE, rmst = TRUE, tau = 18)

# Single-group object: pass group = NULL (or a single-level group)
fit1 <- kmcurve_fast(time[group == 0], event[group == 0])
plot(fit1, rmst = TRUE, tau = 18)
```

maxcombo_fast

Fast Max-Combo Weighted Log-Rank Test for Two-Group Survival Data

Description

Computes the max-combo test, the maximum over a set of Fleming-Harrington weighted log-rank statistics, for comparing survival between two groups under non-proportional hazards. The C++ backend evaluates every weighted numerator and the full between-scheme covariance matrix in a single scan over the pooled sorted data, and the p-value is obtained from the multivariate normal distribution implied by the correlation of the component statistics. The test is robust to the shape of the hazard difference because the most extreme of several complementary weights is taken, with the multiplicity accounted for through the joint distribution.

Usage

```
maxcombo_fast(
  time,
  event,
  group,
  control,
  side = 2,
  rho = c(0, 0, 1, 1),
  gamma = c(0, 1, 0, 1),
```

```

    presorted = FALSE,
    abseps = 1e-05,
    maxpts = 25000
)

```

Arguments

time	A numeric vector of follow-up times for all subjects.
event	An integer or numeric vector of event indicators (1 = event, 0 = censored), aligned with time.
group	A vector of group labels aligned with time.
control	A scalar value indicating which level of group represents the control group.
side	An integer, either 1 or 2. If side = 1, the one-sided max-combo test for treatment benefit is computed. If side = 2 (default), the two-sided test based on the maximum absolute component is computed.
rho	A numeric vector of Fleming-Harrington first parameters, one per component weight. Defaults to $c(0, 0, 1, 1)$.
gamma	A numeric vector of Fleming-Harrington second parameters, one per component weight, aligned with rho. Defaults to $c(0, 1, 0, 1)$. The default pairs are the standard four-weight max-combo: $G(0,0)$ for proportional hazards, $G(0,1)$ for late differences, $G(1,0)$ for early differences, and $G(1,1)$ for middle differences.
presorted	A logical value. If TRUE, time, event, and group are assumed to be sorted in ascending order of time, and the internal <code>order()</code> call is skipped. If FALSE (default), sorting is handled internally. The order is checked, and an error is given when it does not hold.
abseps	A single positive numeric value, the absolute error tolerance passed to the multivariate normal integration. Defaults to 1e-5. Larger values speed up the quasi-Monte-Carlo integration (four or more weights, or a two-sided test) at the cost of p-value precision.
maxpts	A single positive integer, the maximum number of function evaluations for the quasi-Monte-Carlo integration used with four or more weights or with a two-sided test (see Details). Defaults to 25000.

Details

Each component is a Fleming-Harrington $G(\text{rho}, \text{gamma})$ weighted log-rank statistic $Z_k = U_k / \sqrt{V_{kk}}$, where $U_k = \sum w_k (O_{.1} - E_{.1})$ and the weight is $w_k = S(t)^{\text{rho}_k} (1 - S(t))^{\text{gamma}_k}$ evaluated from the left-continuous pooled Kaplan-Meier estimate $S(t)$. The between-scheme covariance is $V_{ab} = \sum w_a w_b v$, with v the hypergeometric variance increment shared by all schemes, so the diagonal of V reproduces the single-scheme weighted variances and the off-diagonal entries give the correlation matrix R of the component Z -scores. The sign convention matches `survdiff_fast`: a component Z is negative when the treatment group is favored.

The max-combo statistic and its p-value depend on side. When side = 1, the statistic is the most negative component, $\min_k Z_k$, so that a negative value favors the treatment group in the same way as `survdiff_fast` with side = 1. The one-sided p-value is $1 - P(G_1 \geq m, \dots, G_K \geq m)$ for G distributed as multivariate normal with mean zero and correlation R , where $m = \min_k Z_k$.

When `side = 2`, the statistic is $\max_k \text{abs}(Z_k)$ and the p-value is $1 - P(-m \leq G_1 \leq m, \dots, -m \leq G_K \leq m)$.

The joint normal probability is evaluated by dimension. With a single weight the univariate normal is used. With two or three weights and a one-sided test, where the integration region is a half-space, the deterministic [TVPACK](#) algorithm is used. For the two-sided test, whose region is a bounded rectangle, and for four or more weights, the quasi-Monte-Carlo [GenzBretz](#) algorithm is used, whose precision is governed by `abseps` and `maxpts`. In a simulation study the Monte Carlo error of the estimated rejection rate is driven by the number of simulated trials rather than by the precision of each individual p-value, so `abseps` can be loosened to speed up the four-weight case with negligible effect on the operating characteristics.

When `presorted = TRUE`, the inputs are assumed to be sorted in ascending order of time and the internal `order()` call is skipped, which is useful inside simulation loops where the data are generated in sorted order.

Value

An object of class `"maxcombo_fast"`, a named numeric vector of length two with elements `statistic` (the max-combo statistic; $\min_k Z_k$ when `side = 1`, so a negative value favors treatment, and $\max_k \text{abs}(Z_k)$ when `side = 2`) and `p.value`. The component Z-scores are stored in the attribute `z`, their correlation matrix in `corr`, the Fleming-Harrington parameters in `rho` and `gamma`, the requested side, and the total sample size in `n`. Returns NA values (still with class `"maxcombo_fast"`) when any component variance is zero or not finite.

References

Karrison, T. G. (2016). Versatile tests for comparing survival curves based on weighted log-rank statistics. *The Stata Journal*, 16(3), 678-690.

Lin, R. S., Lin, J., Royston, S., et al. (2020). Alternative analysis methods for time to event endpoints under nonproportional hazards: a comparative analysis. *Statistics in Biopharmaceutical Research*, 12(2), 187-198.

See Also

[survdif_fast](#) for the single-scheme weighted log-rank test.

Examples

```
library(survival)

# Standard four-weight max-combo, one-sided
fit <- maxcombo_fast(ovarian$futime, ovarian$fustat, ovarian$rx, control = 1,
                    side = 1)
fit["statistic"]
fit["p.value"]

# Two-sided test
maxcombo_fast(ovarian$futime, ovarian$fustat, ovarian$rx, 1, side = 2)

# Custom weight set: proportional plus late-difference only
```

```

maxcombo_fast(ovarian$futime, ovarian$fustat, ovarian$rx, 1,
              rho = c(0, 0), gamma = c(0, 1))

# presorted = TRUE: sort once outside, reuse inside a loop
ord <- order(ovarian$futime)
maxcombo_fast(ovarian$futime[ord], ovarian$fustat[ord], ovarian$rx[ord],
              control = 1, presorted = TRUE)

# Cross-check against simtrial::maxcombo (one-sided)
if (requireNamespace("simtrial", quietly = TRUE)) {
  df <- data.frame(
    stratum = "All",
    treatment = ifelse(ovarian$rx == 2, "experimental", "control"),
    tte = ovarian$futime,
    event = ovarian$fustat
  )
  simtrial::maxcombo(df, rho = c(0, 0, 1, 1), gamma = c(0, 1, 0, 1),
                    return_corr = TRUE)$p_value
}

```

medsurv_fast

Fast non-parametric median survival time and between-group difference

Description

Estimates the Kaplan-Meier median survival time for a single group, or the difference in median survival time between a treatment group and a control group, together with standard errors, confidence intervals and a Wald test for the difference. The estimator is designed for repeated evaluation inside simulation loops, with the single scan over the sorted data performed in C++.

Usage

```

medsurv_fast(
  time,
  event,
  group = NULL,
  control = NULL,
  side = 2,
  conf.level = 0.95,
  conf.type = "log",
  method = c("km", "nph"),
  bw = NULL,
  presorted = FALSE
)

```

Arguments

time	Numeric vector of event or censoring times.
event	Integer vector, 1 for an event and 0 for censoring.
group	Optional grouping vector with exactly two distinct levels for a two-group comparison. If omitted, a single-group median is returned.
control	The level of group that denotes the control group. Required when group is supplied.
side	Either 2 for a two-sided test or 1 for a one-sided test of treatment superiority (difference greater than 0).
conf.level	Confidence level for the intervals.
conf.type	Confidence interval type for each group median, either "plain" or "log". The interval for the difference is always on the plain scale.
method	Variance method, either "km" (Greenwood increment with a kernel hazard) or "nph" (local constant hazard with a $\sum 1 / (Y - k)^2$ increment, matching <code>nph : : nphparams</code>).
bw	Optional kernel bandwidth for the hazard at the median, used only when method = "km". Either a single value applied to both groups or one value per group. If omitted, a Silverman type default is used.
presorted	Logical; set to TRUE when time, event and group are already sorted in ascending order of time, to skip the internal sort. Defaults to FALSE. The order is checked, and an error is given when it does not hold.

Details

The median in each group is the first event time at which the Kaplan-Meier estimate drops to 0.5 or below. When the estimate equals 0.5 exactly over an interval, the midpoint of that interval is returned (the next event time, or the last observed time when no later event exists, closes the interval). This is the convention used by `survfit`, including its tolerance of `sqrt(.Machine$double.eps)` when comparing the estimate with 0.5. The point estimate is the same for both variance methods.

Two variance methods are available through the `method` argument. With `method = "km"` the variance of the estimated median follows the counting process delta method, `var(median) = greenwood_sum / hazard^2`, where the instantaneous hazard at the median is obtained by a Ramlau-Hansen kernel smoother with an Epanechnikov kernel and bandwidth `bw`. With `method = "nph"` the variance reproduces the computation used by `nph : : nphparams` with `param_type = "Q"` and `haz_method = "local"`: a local constant hazard at the median and a $\sum 1 / (Y - k)^2$ variance increment. The two methods give the same median but generally different standard errors, since the variance of a quantile depends on the local hazard estimate. When the Kaplan-Meier and Nelson-Aalen medians coincide, which is the usual case, `method = "nph"` reproduces the `nph : : nphparams` standard error and p-value to numerical precision.

The difference is computed as treatment minus control, so a positive difference indicates a longer median survival time under treatment.

Value

A named numeric vector of class `"medsurv_fast"`. For a single group the elements are the median, its standard error and confidence limits. For two groups the elements are the control and treatment

medians, their difference, the standard errors, the confidence limits and the Wald statistics for the difference.

References

Fleming, T. R., & Harrington, D. P. (1991). *Counting Processes and Survival Analysis*. New York: John Wiley & Sons.

Examples

```
set.seed(1)
n <- 200
g <- rep(0:1, each = n / 2)
tt <- rexp(n, rate = ifelse(g == 0, 0.1, 0.07))
cc <- rexp(n, rate = 0.02)
time <- pmin(tt, cc)
event <- as.integer(tt <= cc)
medsurv_fast(time, event, group = g, control = 0)
medsurv_fast(time, event, group = g, control = 0, method = "nph")
```

milestone_fast

Compare Milestone Survival Probabilities Between Two Groups

Description

Compares the Kaplan-Meier survival probabilities of two groups at a prespecified milestone time-point. The point estimate of interest is the difference in milestone survival, treatment minus control. Three inference methods are provided. The "wald" method uses the unpooled Greenwood variance directly. The "loglog" and "mover" methods build the confidence interval for the difference with the method of variance estimates recovery (MOVER), recovering the variance from the one-sample complementary log-log and log transformed confidence intervals respectively. When the estimate of a group is 0 or 1 at tau, its one-sample interval degenerates to the estimate, so the interval of the difference is still given; the "loglog" test statistic is then NA. See Tang (2021) for the MOVER difference interval and Tang (2022) for the use of milestone survival in trial design.

Usage

```
milestone_fast(
  time,
  event,
  group,
  control,
  side = 2,
  conf.level = 0.95,
  tau,
  method = c("wald", "loglog", "mover"),
  presorted = FALSE
)
```

Arguments

time	A numeric vector of follow-up times.
event	An integer vector of event indicators, 1 for an event and 0 for a censored observation.
group	A vector with exactly two distinct values identifying the group.
control	The value of group that denotes the control group. The other value is the treatment group and the difference is reported as treatment minus control.
side	1 for a one-sided test in the direction of treatment benefit (treatment milestone survival larger than control) or 2 for a two-sided test (default 2). The confidence interval is always reported as a two-sided interval at <code>conf.level</code> .
conf.level	The confidence level for the reported intervals.
tau	The milestone timepoint at which the survival probabilities are compared. A single positive number. A warning is given when it exceeds the largest observed time of either group, because the Kaplan-Meier curve is not estimated beyond that time.
method	The inference method for the difference in milestone survival, one of "wald", "loglog", or "mover".
presorted	Logical. If TRUE the input is assumed to be sorted by time in ascending order and the internal sort is skipped. This is intended for repeated calls inside simulation loops. The order is checked, and an error is given when it does not hold.

Value

An object of class "milestone_fast", a list with the per-group milestone survival estimates and standard errors, the difference estimate with its confidence interval, the test statistic, and the p-value.

References

Tang, Y. (2021). Some new confidence intervals for Kaplan-Meier based estimators from one and two sample survival data. *Statistics in Medicine*, 40(23), 4961-4976.

Tang, Y. (2022). Complex survival trial design by the product integration method. *Statistics in Medicine*, 41(4), 798-814.

Examples

```
set.seed(1)
time <- c(rexp(50, 0.1), rexp(50, 0.07))
event <- rep(1, 100)
group <- rep(c(0, 1), each = 50)
milestone_fast(time, event, group, control = 0, tau = 10, method = "loglog")
```

pairwise_fast	<i>Pairwise comparisons of each experimental arm against a shared control</i>
---------------	---

Description

Runs the two-group analysis of [analysis_fast](#) once for each experimental arm against a common control arm, on multi-arm simulated data such as the output of [simdata_fast](#) with a length-greater-than-two `n`. The contrasts are analyzed at a shared set of looks, and the results are stacked into one long data frame with an `arm` column, optionally with a Bonferroni-adjusted p-value across the contrasts.

Usage

```
pairwise_fast(
  data,
  control,
  event.looks = NULL,
  time.looks = NULL,
  primary = NULL,
  arms = NULL,
  stat = "logrank",
  adjust = c("none", "bonferroni"),
  p.col = NULL,
  cutoff.looks = NULL,
  ...
)
```

Arguments

<code>data</code>	A data frame of simulated trial data with columns <code>sim</code> , <code>group</code> , <code>accrual_time</code> , <code>tte</code> , and <code>event</code> , as produced by simdata_fast .
<code>control</code>	The group label of the control arm.
<code>event.looks</code>	A numeric vector of target cumulative event counts for the primary contrast, one per look. Mutually exclusive with <code>time.looks</code> and <code>cutoff.looks</code> ; requires <code>primary</code> .
<code>time.looks</code>	A numeric vector of calendar times, one per look. Mutually exclusive with <code>event.looks</code> and <code>cutoff.looks</code> .
<code>primary</code>	The group label of the experimental arm whose control-versus-arm comparison defines the shared calendar cutoff. Required with <code>event.looks</code> and ignored with <code>time.looks</code> and <code>cutoff.looks</code> .
<code>arms</code>	A vector of experimental arm labels to compare against the control. Defaults to every group other than control.
<code>stat</code>	The test statistic passed to analysis_fast (for example "logrank", "coxph", or "rmst").

<code>adjust</code>	Multiplicity adjustment across contrasts. Either "none" (default) or "bonferroni", which adds a <code>p.adj</code> column.
<code>p.col</code>	The name of the p-value column to adjust. By default it is the column matching the chosen <code>stat</code> (for example "logrank.p").
<code>cutoff.looks</code>	A numeric matrix of per-simulation calendar cutoffs with one row per simulated trial and one column per look, as returned by <code>cutoff_fast</code> . Mutually exclusive with <code>event.looks</code> and <code>time.looks</code> . See <code>analysis_fast</code> for how the rows are matched to the simulations.
<code>...</code>	Further arguments passed to <code>analysis_fast</code> , such as <code>side</code> , <code>conf.level</code> , <code>tau</code> , <code>rho</code> , and <code>gamma</code> . The argument by <code>subgroup</code> is not supported.

Details

Three timing regimes are supported, exactly one of which must be requested.

With `time.looks`, every contrast is analyzed at the same fixed calendar time or times, so all contrasts share the same data cutoff by construction.

With `event.looks`, the analysis is event-driven and `primary` names the experimental arm whose control-versus-arm comparison defines the data cutoff. The primary contrast is analyzed at the requested cumulative event counts, its per-simulation calendar cutoffs are recorded, and then every contrast, including the primary one, is analyzed at those same cutoffs. This reproduces the standard design in which the primary event-driven analysis fixes a single data cutoff at which all comparisons are performed. For a simulation in which the primary event target is not reached at a look, that look is marked with `reached = FALSE`, and its counts and statistics are NA for every contrast. The shared cutoffs are computed by `cutoff_fast` on the events of the control and primary arms.

With `cutoff.looks`, every contrast is analyzed at the supplied per-simulation calendar cutoffs, for example those returned by `cutoff_fast` for a combination of event and calendar-time rules. A look that is not reached in a simulation (NA cutoff) is marked with `reached = FALSE` and NA counts and statistics, as in the event-driven regime. This differs from `analysis_fast`, which analyzes the full data at an unreached look; in `simsummary_fast` an unreached look of `pairwise_fast` output therefore never crosses a boundary.

The Bonferroni option multiplies each p-value by the number of contrasts and caps it at one, controlling the family-wise error rate across the control-versus-arm comparisons at each look. Multiplicity across looks is a separate matter handled by group-sequential boundaries in `simsummary_fast`, not by this adjustment; given the output of this function, `simsummary_fast` summarizes each arm separately (the Bonferroni-adjusted p-values can be used there through `p.col = "p.adj"`).

This is a single-endpoint helper: it reads the `tte` and `event` columns and does not support subgroups. Comparisons for a second endpoint are obtained by calling `pairwise_fast` again on that endpoint's columns.

Value

A data frame with one row per contrast, look, and simulation. The first column is `arm` (the experimental arm compared against control), followed by the columns returned by `analysis_fast`: `sim`, `look`, `look.value`, `cutoff`, `reached`, the enrollment and event counts, and the statistic columns for the chosen `stat`. When `adjust = "bonferroni"` a `p.adj` column is appended.

See Also

[simdata_fast](#), [analysis_fast](#), [cutoff_fast](#)

Examples

```
# Three-arm trial: control (group 1) and two experimental arms.
dfk <- simdata_fast(
  nsim      = 100,
  n         = c(120, 120, 120),
  a.time    = c(0, 12),
  a.rate    = 360 / 12,
  e.median  = list(12, 16, 20),
  seed      = 8
)

# Fixed calendar look at month 30, Bonferroni across the two contrasts.
pw <- pairwise_fast(dfk, control = 1, time.looks = 30,
  stat = "logrank", side = 1, adjust = "bonferroni")
head(pw)

# Event-driven: arm 3 is the primary contrast and its 200th control-plus-arm-3
# event fixes the cutoff at which both contrasts are analyzed.
pw2 <- pairwise_fast(dfk, control = 1, event.looks = 200, primary = 3,
  stat = "logrank", side = 1, adjust = "bonferroni")
head(pw2)
```

plot.kmcurve_fast

Plot Method for Kaplan-Meier Curves

Description

Draws the Kaplan-Meier curves of a `kmcurve_fast` object. With `rmst = TRUE` the area under each curve up to `tau` is shaded to visualize the restricted mean survival time. With `hr = TRUE` a second panel shows the smoothed time-varying hazard ratio together with the constant Cox proportional-hazards estimate and, optionally, the average hazard ratio, in the style of stylized nonproportional-hazards figures. For a single-group object only one survival curve is drawn; `hr = TRUE` is ignored because a hazard ratio requires two groups.

Usage

```
## S3 method for class 'kmcurve_fast'
plot(
  x,
  hr = FALSE,
  rmst = FALSE,
  tau = NULL,
  bw = NULL,
```

```

    n_grid = 200L,
    conf.int = TRUE,
    hr_max = NULL,
    cox_line = TRUE,
    ahr_line = FALSE,
    mark_time = TRUE,
    conf.level = 0.95,
    conf.type = "log",
    col_control = "#1b9e77",
    col_treat = "#d95f02",
    col_hr = "#3060c0",
    col_cox = "black",
    col_ahr = "grey40",
    lty_control = 1,
    lty_treat = 1,
    lty_hr = 1,
    lwd_control = 2,
    lwd_treat = 2,
    lwd_hr = 2,
    base_size = 1,
    legend_pos = "topright",
    xlab = "Time",
    ylab = "Survival probability",
    main_km = "Kaplan-Meier",
    main_hr = "Hazard ratio",
    group_labels = NULL,
    ...
  )

```

Arguments

x	A kmcurve_fast object from kmcurve_fast .
hr	Logical. If TRUE, add a hazard-ratio panel to the right.
rmst	Logical. If TRUE, shade the area under each curve up to tau and annotate the restricted mean survival time difference.
tau	Upper limit used for the restricted mean survival time and the average hazard ratio. Defaults to the smaller of the two groups' largest observed times.
bw	Kernel bandwidth for the smoothed hazard ratio. Defaults to a value derived from the spread of the event times.
n_grid	Number of grid points for the smoothed hazard ratio.
conf.int	Logical. If TRUE, add pointwise confidence limits to the survival curves.
hr_max	Upper limit of the hazard-ratio axis. Defaults to a value derived from the estimated hazard ratio.
cox_line	Logical. If TRUE, add the constant Cox hazard ratio as a horizontal line in the hazard-ratio panel.
ahr_line	Logical. If TRUE, add the average hazard ratio as a horizontal line in the hazard-ratio panel.

mark_time	Logical. If TRUE, mark censoring times on the curves.
conf.level	Confidence level for the survival confidence limits.
conf.type	Type of survival confidence limits: one of "plain", "log" (default), or "log-log", matching survfit_fast .
col_control, col_treat, col_hr, col_cox, col_ahr	Colors for the control curve, the treatment curve, the smoothed hazard ratio, the Cox line, and the average-hazard-ratio line.
lty_control, lty_treat	Line types for the control and treatment curves.
lty_hr	Line type for the smoothed hazard ratio.
lwd_control, lwd_treat	Line widths for the control and treatment curves.
lwd_hr	Line width for the lines in the hazard-ratio panel.
base_size	Relative size of all text.
legend_pos	Keyword giving the legend position in both panels, passed to legend (for example "topright" or "bottomleft").
xlab, ylab	Axis labels for the survival panel.
main_km, main_hr	Panel titles.
group_labels	Optional length-two character vector of legend labels for the control and treatment groups.
...	Further arguments, currently ignored.

Details

The smoothed hazard ratio is obtained by kernel smoothing the Nelson-Aalen increments of each group. For group j the kernel hazard estimate is the sum over event times of $(1 / bw) K((t - t_i) / bw) (d_i / Y_i)$, with an Epanechnikov kernel K , and the hazard ratio is the treatment estimate divided by the control estimate. The bandwidth bw controls the width of the averaging window and materially affects the result, so it is exposed as an argument. To limit boundary bias the hazard ratio is drawn over an interior time window, and points where the control hazard estimate is effectively zero are dropped.

The constant Cox hazard ratio is taken from [coxph_fast](#), the average hazard ratio from [ahr_fast](#), and the restricted mean survival time difference from [rmst_fast](#).

Value

The input x , invisibly.

See Also

[kmcurve_fast](#), [print.kmcurve_fast](#)

plot.scenario_fast *Plot Method for Scenario Sets*

Description

Draws, for each scenario in a `scenario_fast` object, the analytic control and experimental survival curves together with the piecewise-constant hazard ratio on a secondary axis, in the style of the stylized nonproportional-hazards figures used in the design literature. The curves are computed analytically from the piecewise-exponential parameters, so no simulation is involved.

Usage

```
## S3 method for class 'scenario_fast'
plot(
  x,
  which = NULL,
  tmax = NULL,
  hr_max = NULL,
  n_time = 481L,
  mfrow = NULL,
  col_control = "#1b9e77",
  col_treat = "#d95f02",
  col_hr = "#3060c0",
  lty_control = 1,
  lty_treat = 1,
  lty_hr = 2,
  lwd_control = 2,
  lwd_treat = 2,
  lwd_hr = 1.5,
  base_size = 1,
  xlab = "Time",
  ylab = "Survival probability",
  legend_panel = 1L,
  legend_pos = "topright",
  ...
)
```

Arguments

<code>x</code>	A <code>scenario_fast</code> object from gen_scenario_fast .
<code>which</code>	Optional integer or character vector selecting a subset of scenarios to draw.
<code>tmax</code>	Upper limit of the time axis. Defaults to a value derived from the scenarios.
<code>hr_max</code>	Upper limit of the hazard-ratio axis. Defaults to a value derived from the scenarios.
<code>n_time</code>	Number of points on the time grid.

<code>mfrow</code>	Optional length-two panel layout <code>c(rows, cols)</code> . Defaults to a near-square grid.
<code>col_control, col_treat, col_hr</code>	Colors for the control survival curve, the experimental survival curve, and the hazard ratio.
<code>lty_control, lty_treat, lty_hr</code>	Line types for the control survival curve, the experimental survival curve, and the hazard ratio.
<code>lwd_control, lwd_treat, lwd_hr</code>	Line widths for the control survival curve, the experimental survival curve, and the hazard ratio.
<code>base_size</code>	Relative size of all text (titles, axis labels, axis annotation, and the legend). The default 1 reproduces the standard sizes; values above one enlarge the text and values below one shrink it.
<code>xlab, ylab</code>	Axis labels for the survival panels.
<code>legend_panel</code>	Index (within the drawn subset) of the panel that carries the legend.
<code>legend_pos</code>	Keyword giving the legend position, passed to legend (for example "topright" or "bottomleft").
<code>...</code>	Further arguments, currently ignored.

Details

Each scenario is shown in its own panel arranged on a grid. The hazard ratio is rescaled into the survival plotting region and read on the right-hand axis; a reference line marks a hazard ratio of one. A single-arm scenario (no treatment specification) is drawn as a single survival curve with no hazard ratio.

Value

The input `x`, invisibly.

See Also

[gen_scenario_fast](#), [print.scenario_fast](#)

<code>print.ahr_fast</code>	<i>Print an <code>ahr_fast</code> object</i>
-----------------------------	--

Description

Print an `ahr_fast` object

Usage

```
## S3 method for class 'ahr_fast'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	an object of class "ahr_fast"
digits	number of significant digits to print
...	further arguments (currently ignored)

Value

x, invisibly

print.ahsw_fast	<i>Print Method for ahsw_fast Objects</i>
-----------------	---

Description

Formats and prints an ahsw_fast object. The header shows the truncation time and the control label. The body shows the per-group average hazard with survival weight, followed by the between-group contrasts: the ratio of average hazards (treatment over control) and the difference of average hazards (treatment minus control), each with a confidence interval and a two-sided p-value.

Usage

```
## S3 method for class 'ahsw_fast'
print(x, digits = max(1L, getOption("digits") - 3L), ...)
```

Arguments

x	An object of class "ahsw_fast" returned by ahsw_fast .
digits	Number of significant digits to display. Defaults to the global option <code>getOption("digits")</code> .
...	Additional arguments (currently unused).

Value

Invisibly returns x.

See Also

[ahsw_fast](#)

Examples

```
library(survival)
fit <- ahsw_fast(ovarian$futime, ovarian$fustat, ovarian$rx,
                 control = 1, tau = 600)
print(fit)
```

print.coxph_fast	<i>Print Method for coxph_fast Objects</i>
------------------	--

Description

Formats and prints a coxph_fast object similarly to `summary(survival::coxph(...))`, showing the point estimate of the log hazard ratio, the hazard ratio, the standard error on the log scale, the Wald z-statistic, the corresponding two-sided p-value, and the Wald confidence interval for the hazard ratio.

Usage

```
## S3 method for class 'coxph_fast'
print(x, digits = max(1L, getOption("digits") - 3L), ...)
```

Arguments

x	An object of class "coxph_fast" returned by coxph_fast .
digits	Number of significant digits to display. Defaults to the global option <code>getOption("digits")</code> .
...	Additional arguments passed to format (currently unused).

Value

Invisibly returns x.

See Also

[coxph_fast](#)

Examples

```
library(survival)
fit <- coxph_fast(ovarian$futime, ovarian$fustat, ovarian$rx, control = 1)
print(fit)
```

print.kmcurve_fast	<i>Print Method for Kaplan-Meier Curves</i>
--------------------	---

Description

Prints a compact per-group summary of a kmcurve_fast object: the group label and role, the sample size, the number of events, and the median survival read from the Kaplan-Meier curve with the convention of `survival::survfit`. In the two-group case the constant Cox proportional-hazards estimate of the hazard ratio is reported below the table.

Usage

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

Arguments

x A kmcurve_fast object from [kmcurve_fast](#).
 ... Further arguments, currently ignored.

Value

The input x, invisibly.

See Also

[kmcurve_fast](#), [plot.kmcurve_fast](#)

print.maxcombo_fast *Print Method for maxcombo_fast Objects*

Description

Formats and prints a maxcombo_fast object. It shows the total sample size, a table of the Fleming-Harrington component Z-scores, and the max-combo statistic with its p-value. For a one-sided test the statistic is the most negative component, so a negative value favors the treatment group, matching the sign convention of [survdiff_fast](#). For a two-sided test the statistic is the largest component in absolute value.

Usage

```
## S3 method for class 'maxcombo_fast'
print(x, digits = max(1L, getOption("digits") - 3L), ...)
```

Arguments

x An object of class "maxcombo_fast" returned by [maxcombo_fast](#).
 digits Number of significant digits to display. Defaults to the global option `getOption("digits")`.
 ... Additional arguments (currently unused).

Value

Invisibly returns x.

See Also

[maxcombo_fast](#)

Examples

```
library(survival)
fit <- maxcombo_fast(ovarian$futime, ovarian$fustat, ovarian$rx,
                    control = 1, side = 1)
print(fit)
```

```
print.medsurv_fast      Print Method for medsurv_fast Objects
```

Description

Formats and prints a `medsurv_fast` object in the same layout as the other two-group estimation summaries in the package. The header shows the control label and the inference settings. The body shows the per-group median survival with its confidence interval, followed, for a two-group object, by the difference contrast (treatment minus control) with a confidence interval, the test statistic, and the p-value.

Usage

```
## S3 method for class 'medsurv_fast'
print(x, digits = max(1L, getOption("digits") - 3L), ...)
```

Arguments

<code>x</code>	An object of class "medsurv_fast" returned by medsurv_fast .
<code>digits</code>	Number of significant digits to display. Defaults to the global option <code>getOption("digits")</code> .
<code>...</code>	Additional arguments (currently unused).

Value

Invisibly returns `x`.

See Also

[medsurv_fast](#)

Examples

```
set.seed(1)
time <- c(rexp(50, 0.1), rexp(50, 0.07))
status <- rep(1, 100)
group <- rep(c(0, 1), each = 50)
print(medsurv_fast(time, status, group, control = 0))
```

print.milestone_fast *Print Method for milestone_fast Objects*

Description

Formats and prints a milestone_fast object in the same layout as the other two-group estimation summaries in the package. The header shows the milestone timepoint, the control label, and the inference settings. The body shows the per-group milestone survival with its confidence interval, followed by the difference contrast (treatment minus control) with a confidence interval, the test statistic, and the p-value. The p-value follows the alternative recorded in the object.

Usage

```
## S3 method for class 'milestone_fast'
print(x, digits = max(1L, getOption("digits") - 3L), ...)
```

Arguments

x	An object of class "milestone_fast" returned by milestone_fast .
digits	Number of significant digits to display. Defaults to the global option <code>getOption("digits")</code> .
...	Additional arguments (currently unused).

Value

Invisibly returns x.

See Also

[milestone_fast](#)

Examples

```
set.seed(1)
time <- c(rexp(50, 0.1), rexp(50, 0.07))
status <- rep(1, 100)
group <- rep(c(0, 1), each = 50)
print(milestone_fast(time, status, group, control = 0, tau = 10, method = "loglog"))
```

print.rmst_fast	<i>Print Method for rmst_fast Objects</i>
-----------------	---

Description

Formats and prints an `rmst_fast` object. In single-group mode it shows the restricted mean survival time, its Greenwood standard error, and the Wald confidence interval at the requested horizon. In two-group mode it shows the per-group restricted mean survival times together with the difference (treatment minus control) and ratio (treatment over control) contrasts, each with a Wald z-statistic and two-sided p-value.

Usage

```
## S3 method for class 'rmst_fast'
print(x, digits = max(1L, getOption("digits") - 3L), ...)
```

Arguments

<code>x</code>	An object of class "rmst_fast" returned by rmst_fast .
<code>digits</code>	Number of significant digits to display. Defaults to the global option <code>getOption("digits")</code> .
<code>...</code>	Additional arguments (currently unused).

Value

Invisibly returns `x`.

See Also

[rmst_fast](#)

Examples

```
set.seed(42)
t_raw <- rexp(100, rate = 1 / 10)
e_raw <- rbinom(100, 1, 0.7)

# Single-group
print(rmst_fast(t_raw, e_raw, tau = 10))

# Two-group comparison
set.seed(7)
n <- 200
time <- c(rexp(n, 0.10), rexp(n, 0.07))
event <- rbinom(2 * n, 1, 0.8)
group <- rep(0:1, each = n)
print(rmst_fast(time, event, group = group, control = 0, tau = 10))
```

print.rmw_fast	<i>Print Method for rmw_fast Objects</i>
----------------	--

Description

Formats and prints an `rmw_fast` object, showing the standardized log-rank and modestly-weighted component Z-scores, their null correlation, the survival-probability threshold `s_star`, the combined test statistic, and the corresponding p-value.

Usage

```
## S3 method for class 'rmw_fast'
print(x, digits = max(1L, getOption("digits") - 3L), ...)
```

Arguments

<code>x</code>	An object of class " <code>rmw_fast</code> " returned by rmw_fast .
<code>digits</code>	Number of significant digits to display. Defaults to the global option <code>getOption("digits")</code> .
<code>...</code>	Additional arguments (currently unused).

Value

Invisibly returns `x`.

See Also

[rmw_fast](#)

Examples

```
library(survival)
fit <- rmw_fast(ovarian$futime, ovarian$fustat, ovarian$rx,
               control = 1, side = 1, s_star = 0.5)
print(fit)
```

print.scenario_fast	<i>Print Method for Scenario Sets</i>
---------------------	---------------------------------------

Description

Prints a compact one-row-per-scenario summary of a `scenario_fast` object: the scenario label, the null flag, the total sample size, the control and experimental medians, the hazard ratio at the start and end of the time window, and whether the two survival curves cross. The medians and hazard ratios are read from the analytic survival curves, so they are defined for both exponential and piecewise-exponential scenarios.

Usage

```
## S3 method for class 'scenario_fast'
print(x, tmax = NULL, ...)
```

Arguments

x	A scenario_fast object from gen_scenario_fast .
tmax	Upper limit of the time window used to read medians, the end-of- window hazard ratio, and the crossing flag. Defaults to a value derived from the scenarios.
...	Further arguments, currently ignored.

Value

The input x, invisibly.

See Also

[gen_scenario_fast](#), [plot.scenario_fast](#)

print.simsummary_fast *Print Method for Sequential Analysis Summaries*

Description

Formats an object returned by [simsummary_fast](#) in the style of a group-sequential design report. After a header with the simulation count and the boundary settings, two look-by-look tables are shown: a stopping-boundary table (information fraction, events, sample size, the efficacy and futility boundaries, and the cumulative efficacy crossing probability) and an analysis-timing table (sample size, events, dropouts, pipeline, analysis time, and the per-look efficacy and futility crossing probabilities). An overall block reports the rejection rate and the expected counts and timing at the stopping look.

Usage

```
## S3 method for class 'simsummary_fast'
print(x, digits = 4, ...)
```

Arguments

x	An object of class "simsummary_fast" from simsummary_fast .
digits	A single positive integer, the number of decimal places used for the printed probabilities. Defaults to 4.
...	Further arguments, currently ignored.

Details

Column labels follow the convention of group-sequential design software: Events (s), Sample (n), Dropouts (d), Pipeline (the enrolled count minus events minus dropouts), Analysis Time (mean calendar time), and Info. Frac. (the information fraction, computed as the mean events at a look divided by the mean events at the final look, or from look.value when no event count is available). Probabilities are printed to digits decimal places and counts and times to fewer. Because the summary is a Monte Carlo estimate under a single data-generating truth, it does not carry the separate null and alternative columns or the alpha and beta spending of an analytic design report. When the summary has several blocks (arms of [pairwise_fast](#) output or subgroup populations), each block is printed under its own heading. The underlying object is an ordinary data frame, so the unrounded values remain available by subsetting it directly.

Value

The object x, invisibly.

See Also

[simsummary_fast](#).

print.survdiff_fast *Print Method for survdiff_fast Objects*

Description

Formats and prints a survdiff_fast object similarly to print(survival::survdiff(...)), showing the observed and expected event counts for the control and treatment groups, the per-group contributions $(O-E)^2 / E$ and $(O-E)^2 / V$, the test statistic, and the corresponding p-value. For a weighted log-rank test the header names the weight scheme (and notes stratification), and only the observed event counts are shown alongside the weighted statistic, since a single unweighted expected count is not defined for a weighted test.

Usage

```
## S3 method for class 'survdiff_fast'
print(x, digits = max(1L, getOption("digits") - 3L), ...)
```

Arguments

x	An object of class "survdiff_fast" returned by survdiff_fast .
digits	Number of significant digits to display. Defaults to the global option getOption("digits").
...	Additional arguments (currently unused).

Value

Invisibly returns x.

See Also[survdiff_fast](#)**Examples**

```
library(survival)
fit <- survdiff_fast(ovarian$futime, ovarian$fustat, ovarian$rx,
                    control = 1, side = 2)
print(fit)
```

print.survfit_fast	<i>Print Method for survfit_fast Objects</i>
--------------------	--

Description

Formats and prints a `survfit_fast` object similarly to `print(summary(survival::survfit(...), times = t_eval))`, showing the Kaplan-Meier survival estimate, the Greenwood standard error on the survival scale, and the confidence interval at the requested evaluation time.

Usage

```
## S3 method for class 'survfit_fast'
print(x, digits = max(1L, getOption("digits") - 3L), ...)
```

Arguments

<code>x</code>	An object of class "survfit_fast" returned by survfit_fast .
<code>digits</code>	Number of significant digits to display. Defaults to the global option <code>getOption("digits")</code> .
<code>...</code>	Additional arguments (currently unused).

Value

Invisibly returns `x`.

See Also[survfit_fast](#)**Examples**

```
set.seed(42)
t_raw <- rexp(100, rate = 1 / 10)
e_raw <- rbinom(100, 1, 0.7)
ord <- order(t_raw)
fit <- survfit_fast(t_raw[ord], e_raw[ord], t_eval = 10)
print(fit)
```

print.wkm_fast*Print Method for wkm_fast Objects*

Description

Formats and prints a `wkm_fast` object in the same layout as the other two-group summaries in the package. The header shows the control label and the inference settings. The body shows the weighted integrated survival difference (treatment minus control) with a confidence interval, the test statistic, and the p-value.

Usage

```
## S3 method for class 'wkm_fast'
print(x, digits = max(1L, getOption("digits") - 3L), ...)
```

Arguments

<code>x</code>	An object of class "wkm_fast" returned by wkm_fast .
<code>digits</code>	Number of significant digits to display. Defaults to the global option <code>getOption("digits")</code> .
<code>...</code>	Additional arguments (currently unused).

Value

Invisibly returns `x`.

See Also

[wkm_fast](#)

Examples

```
set.seed(1)
n <- 200
g <- rep(0:1, each = n / 2)
tt <- c(rexp(n / 2, 0.1), rexp(n / 2, 0.07))
cc <- rexp(n, 0.02)
time <- pmin(tt, cc)
event <- as.integer(tt <= cc)
print(wkm_fast(time, event, group = g, control = 0))
```

`print.wmst_fast`*Print Method for wmst_fast Objects*

Description

Formats and prints a `wmst_fast` object in the same layout as the other two-group summaries in the package. The header shows the window, the control label and the inference settings. The body shows the per-group window mean survival time with its confidence interval, followed, for a two-group object, by the difference contrast (treatment minus control) with a confidence interval, the test statistic, and the p-value.

Usage

```
## S3 method for class 'wmst_fast'
print(x, digits = max(1L, getOption("digits") - 3L), ...)
```

Arguments

<code>x</code>	An object of class "wmst_fast" returned by wmst_fast .
<code>digits</code>	Number of significant digits to display. Defaults to the global option <code>getOption("digits")</code> .
<code>...</code>	Additional arguments (currently unused).

Value

Invisibly returns `x`.

See Also

[wmst_fast](#)

Examples

```
set.seed(1)
n <- 200
g <- rep(0:1, each = n / 2)
tt <- c(rexp(n / 2, 0.1), rexp(n / 2, 0.07))
cc <- rexp(n, 0.02)
time <- pmin(tt, cc)
event <- as.integer(tt <= cc)
print(wmst_fast(time, event, group = g, control = 0, tau1 = 2, tau2 = 12))
```

rmst_fast	<i>Fast Restricted Mean Survival Time (Single Group or Two-Group Comparison)</i>
-----------	--

Description

Computes the restricted mean survival time (RMST) up to a horizon τ from the Kaplan-Meier estimator. With a single group (the default), it returns the RMST with its Greenwood-type standard error and a Wald confidence interval. When a group is supplied, it additionally returns the two-group contrasts: the RMST difference (treatment minus control) and the RMST ratio (treatment over control), each with a standard error, confidence interval, and test (one- or two-sided, following `side`). The C++ backend integrates the survival step function in a single scan and is reused once per group, so the function is suitable for simulation loops with `presorted = TRUE`.

Usage

```
rmst_fast(
  time,
  event,
  group = NULL,
  control = NULL,
  side = 2,
  conf.level = 0.95,
  tau,
  presorted = FALSE
)
```

Arguments

<code>time</code>	A numeric vector of event or censoring times.
<code>event</code>	An integer or numeric vector of event indicators (1 = event, 0 = censored), aligned with <code>time</code> .
<code>group</code>	An optional vector of group labels aligned with <code>time</code> . If <code>NULL</code> (default), a single-group RMST is computed. If supplied, the two-group contrasts are returned and <code>control</code> must be given.
<code>control</code>	A scalar value indicating which level of group represents the control group. Required when <code>group</code> is supplied.
<code>side</code>	1 for a one-sided test in the direction of treatment benefit (longer restricted mean survival in the treatment group) or 2 for a two-sided test (default 2). Applies to the two-group contrasts; the confidence intervals are always two-sided at <code>conf.level</code> .
<code>conf.level</code>	A single numeric value in (0, 1) specifying the confidence level. Defaults to 0.95.
<code>tau</code>	A single positive numeric value specifying the restriction horizon. A warning is given when it exceeds the largest observed time (of either group in the two-group case), because the Kaplan-Meier curve is not estimated beyond that time.

presorted A logical value. If TRUE, the inputs are assumed to be sorted in ascending order of time. If FALSE (default), sorting is handled internally. The order is checked, and an error is given when it does not hold.

Details

The RMST is the area under the Kaplan-Meier curve from 0 to tau:

$RMST(\tau) = \text{integral over } [0, \tau] \text{ of } S(u) \, du.$

The variance follows the Greenwood-type estimator

$Var[RMST] = \sum_{t_i \leq \tau} A_i^2 d_i / (n_i (n_i - d_i)),$

where $A_i = \text{integral over } [t_i, \tau] \text{ of } S(u) \, du$ is the area to the right of event time t_i . This matches the restricted mean reported by [survfit](#) and by the `survRM2` package.

When `group` is supplied, the treatment group is the level of `group` that is not equal to control. Writing the treatment and control RMST as r_1 and r_0 with variances v_1 and v_0 , the difference contrast is $\text{diff} = r_1 - r_0$ with $Var[\text{diff}] = v_1 + v_0$, using the independence of the two groups. The ratio contrast is formed on the log scale by the delta method, $Var[\log(r_1 / r_0)] = v_1 / r_1^2 + v_0 / r_0^2$, with the confidence interval exponentiated back to the ratio scale. These match the unadjusted contrasts reported by `survRM2`.

When `presorted = TRUE`, the input vectors are assumed to be sorted in ascending order of time; splitting into groups preserves the ascending order within each group, so no re-sorting is performed. When `presorted = FALSE` (default), sorting is handled internally. In simulation loops where the data are generated in sorted order, `presorted = TRUE` avoids one $O(n \log n)$ pass.

Value

An object of class "rmst_fast", a named numeric vector. In single-group mode it has length 4 with elements `rmst`, `std.err`, `lower`, and `upper`. In two-group mode it contains the per-group RMST (`rmst.ctrl`, `rmst.trt`), the difference contrast (`diff`, `se.diff`, `diff.lower`, `diff.upper`, `z.diff`, `p.diff`), and the ratio contrast (`ratio`, `ratio.lower`, `ratio.upper`, `z.ratio`, `p.ratio`). The restriction horizon, confidence level, and test side are stored as attributes `tau`, `conf.level`, and `side`; in two-group mode the control label is also stored. Returns `NA_real_` values (still with class "rmst_fast") when n is zero in single-group mode.

References

Royston, P., & Parmar, M. K. B. (2013). Restricted mean survival time: an alternative to the hazard ratio for the design and analysis of randomized trials with a time-to-event outcome. *BMC Medical Research Methodology*, 13, 152.

Uno, H., Claggett, B., Tian, L., et al. (2014). Moving beyond the hazard ratio in quantifying the between-group difference in survival analysis. *Journal of Clinical Oncology*, 32(22), 2380-2385.

See Also

[survfit_fast](#) for the Kaplan-Meier estimate at a single time point.

Examples

```

set.seed(42)
t_raw <- rexp(100, rate = 1 / 10)
e_raw <- rbinom(100, 1, 0.7)

# Single-group RMST
rmst_fast(t_raw, e_raw, tau = 10)

# Single-group, pre-sorted (sort once, reuse in a loop)
ord <- order(t_raw)
rmst_fast(t_raw[ord], e_raw[ord], tau = 10, presorted = TRUE)

# Two-group comparison (difference and ratio)
set.seed(7)
n <- 200
time <- c(rexp(n, 0.10), rexp(n, 0.07))
event <- rbinom(2 * n, 1, 0.8)
group <- rep(0:1, each = n)
rmst_fast(time, event, group = group, control = 0, tau = 10)

# Validation against survRM2
if (requireNamespace("survRM2", quietly = TRUE)) {
  survRM2::rmst2(time, event, group, tau = 10)$unadjusted.result
}

```

rmw_fast

Robust Modestly-Weighted Log-Rank Test for Two-Group Survival Data

Description

Computes the robust modestly-weighted (rMW) log-rank test of Magirr and Ohn, which combines the standard log-rank test with a single modestly-weighted log-rank test. The test statistic is the maximum of the two standardized components, evaluated against their joint null distribution. Because the standard log-rank statistic is included as one of the two components and the components are strongly correlated under the null, the multiplicity adjustment is small, so the test loses little power relative to the log-rank test in worst-case scenarios while gaining substantial power under delayed effects. The two component numerators, their variances, and their null covariance are computed in a single pass by the C++ core `rmw_core`.

Usage

```

rmw_fast(
  time,
  event,
  group,

```

```

    control,
    side = 2,
    s_star = 0.5,
    presorted = FALSE
)

```

Arguments

<code>time</code>	A numeric vector of follow-up times for all subjects.
<code>event</code>	An integer or numeric vector of event indicators (1 = event, 0 = censored), aligned with <code>time</code> .
<code>group</code>	A vector of group labels aligned with <code>time</code> .
<code>control</code>	A scalar value indicating which level of group represents the control group.
<code>side</code>	An integer, either 1 or 2 (default 2). If <code>side = 1</code> , the statistic is the minimum of the two standardized components and a one-sided p-value is returned. If <code>side = 2</code> , the statistic is the maximum of their absolute values and a two-sided p-value is returned.
<code>s_star</code>	A single numeric value in $(0, 1]$, the survival-probability threshold of the modestly-weighted component. The weight is capped at $1 / s_star$. Defaults to 0.5. A value of 1 caps the weight at one, so the modestly-weighted component equals the log-rank component and the test reduces to the ordinary log-rank test.
<code>presorted</code>	A logical value. If TRUE, <code>time</code> , <code>event</code> , and <code>group</code> are assumed to be already sorted by ascending time, and the internal <code>order()</code> call is skipped. If FALSE (default), sorting is handled internally. The order is checked, and an error is given when it does not hold.

Details

The first component is the ordinary log-rank statistic, with weight one at every event time. The second component is a modestly-weighted log-rank statistic with weight $\min(1 / S(t-), 1 / s_star)$, where $S(t-)$ is the left-continuous pooled Kaplan-Meier estimate just prior to each event time and s_star is a survival-probability threshold. This is the survival-threshold parameterization of the modestly-weighted test of Magirr and Burman, in which the weight is capped at $1 / s_star$. It differs from the timepoint parameterization exposed by `survdifffast()` with `weight = "mwlr_t"`, where the cap is derived from a timepoint t_star . The choice $s_star = 0.5$ caps the weight at 2 and is the value used in the original rMW article.

Writing the two standardized components as Z_{lr} and Z_{mw} , under the null hypothesis of equal survival the pair is asymptotically bivariate normal with zero means, unit variances, and correlation $\rho = C / \sqrt{V_{lr} V_{mw}}$, where C is the covariance of the two numerators returned by the C++ core. When `side = 1` the statistic is $\min(Z_{lr}, Z_{mw})$, so a protective treatment effect (fewer events than expected) yields a small, negative value, and the one-sided p-value is $P(\min(Z_{lr}, Z_{mw}) \leq \text{observed})$ under the joint null. When `side = 2` the statistic is $\max(\text{abs}(Z_{lr}), \text{abs}(Z_{mw}))$ and the two-sided p-value is $P(\max(\text{abs}(Z_{lr}), \text{abs}(Z_{mw})) \geq \text{observed})$. The joint normal probability is evaluated with `mvtnorm::pmvnorm`, using the exact TVPACK algorithm for the one-sided half-space and the deterministic Miwa algorithm for the two-sided rectangle.

When `presorted = TRUE`, the input vectors are assumed to be sorted by ascending time and the `internal.order()` call is skipped, which avoids one $O(n \log n)$ pass in simulation loops where the data are already generated in time order.

Value

An object of class `"rmw_fast"`, a length-two numeric vector `c(statistic, p.value)` with attributes `z` (the named component Z-scores `c(logrank, mwlrt)`), `corr` (the 2 by 2 null correlation matrix of the two components), `s_star`, 01 (the observed number of events in the treatment group), `side`, and `n` (the total sample size). Returns NA statistic and p-value (still with class `"rmw_fast"`) when either component variance is zero (e.g., all events in one group).

References

Magirr, D., & Öhrn, F. (2026). Robust modestly weighted log-rank tests. *Pharmaceutical Statistics*, 25(1), e70066.

Magirr, D., & Burman, C.-F. (2019). Modestly weighted logrank tests. *Statistics in Medicine*, 38(20), 3782-3790.

See Also

[survdiff_fast](#) for the standard and weighted log-rank tests. [print.rmw_fast](#) for the print method.

Examples

```
library(survival)

# One-sided robust modestly-weighted test with s_star = 0.5
fit <- rmw_fast(ovarian$futime, ovarian$fustat, ovarian$rx,
               control = 1, side = 1, s_star = 0.5)
fit

# The log-rank component matches survdiff_fast with weight = "logrank"
z_lr <- as.numeric(survdiff_fast(ovarian$futime, ovarian$fustat, ovarian$rx,
                               control = 1, side = 1))
cat("rmw_fast log-rank component:", attr(fit, "z")["logrank"], "\n")
cat("survdiff_fast log-rank Z   :", z_lr, "\n")

# Two-sided test
rmw_fast(ovarian$futime, ovarian$fustat, ovarian$rx,
        control = 1, side = 2, s_star = 0.5)

# presorted = TRUE: sort once outside, reuse inside a loop
ord <- order(ovarian$futime)
rmw_fast(ovarian$futime[ord], ovarian$fustat[ord], ovarian$rx[ord],
        control = 1, side = 1, s_star = 0.5, presorted = TRUE)
```

simdata_fast

*Fast Simulation of Two-Group Time-to-Event Trial Data***Description**

Simulates time-to-event trial data for one or two groups across many simulated trials, with piecewise accrual, piecewise-exponential survival and dropout, and optional subgroups defined by a prevalence specification. The entire generation pipeline (accrual, survival, dropout, derived columns, and two-group interleaving) runs in a single C++ kernel that materializes the output data frame once, avoiding intermediate R-level vector operations and copies. The random-number stream is consumed in the same order as a per-group reference implementation, so results are reproducible from seed.

Usage

```
simdata_fast(
  nsim = 1000,
  n,
  alloc = c(1, 1),
  a.time,
  a.rate = NULL,
  a.prop = NULL,
  e.hazard = NULL,
  e.median = NULL,
  e.time = NULL,
  d.hazard = NULL,
  d.median = NULL,
  d.time = NULL,
  seed = NULL,
  prevalence = NULL,
  fixed.alloc = FALSE,
  h01.hazard = NULL,
  h01.median = NULL,
  h01.time = NULL,
  h02.hazard = NULL,
  h02.median = NULL,
  h02.time = NULL,
  h12.hazard = NULL,
  h12.median = NULL,
  h12.time = NULL,
  switch.prop = NULL,
  h12.switch.hazard = NULL,
  h12.switch.median = NULL,
  h12.switch.time = NULL,
  switch.clock = "reset",
  stream = NULL
)
```

Arguments

<code>nsim</code>	Number of simulated trials.
<code>n</code>	Either a single total sample size (split by <code>alloc</code>), a length-two vector of per-group sample sizes, or, for a multi-arm trial, a vector of length greater than two giving the per-arm sample sizes (which requires a per-arm <code>e.hazard</code> or <code>e.median</code> list). When <code>n</code> is a per-arm vector, <code>alloc</code> is ignored.
<code>alloc</code>	A length-two allocation ratio (control, treatment), used when <code>n</code> is scalar. Supplying it explicitly requests a two-group simulation (see Details). The total is split in proportion to <code>alloc</code> and any rounding remainder goes to the group with the larger fractional share, so the two group sizes always add up to <code>n</code> .
<code>a.time</code>	A numeric vector of accrual-interval breakpoints.
<code>a.rate</code>	Absolute accrual rates (subjects per unit time), interpreted in one of two ways. With length <code>length(a.time) - 1</code> the accrual period is fully specified and the rates must accrue exactly <code>sum(n)</code> subjects (an inconsistent total is an error). With length <code>length(a.time)</code> the final rate applies to an open last interval whose end time is computed so the total is <code>sum(n)</code> . Supply exactly one of <code>a.rate</code> and <code>a.prop</code> .
<code>a.prop</code>	Accrual proportions, one per accrual interval (length <code>length(a.time) - 1</code>), giving the fraction of subjects enrolled in each interval. Values are normalized to sum to one and distribute the fixed total <code>sum(n)</code> . Unlike <code>a.rate</code> this carries no rate, so the accrual period must be fully specified by <code>a.time</code> . Supply exactly one of <code>a.rate</code> and <code>a.prop</code> .
<code>e.hazard</code>	Survival hazard(s). A scalar or vector (piecewise, with <code>e.time</code>) shared by all groups and cells, a two-element list for two groups, or, with subgroups, a per-cell list (for one group, or as an element of the two-element list). See Details for how the number of groups is determined.
<code>e.median</code>	Survival median(s); an alternative to <code>e.hazard</code> .
<code>e.time</code>	Survival breakpoints for piecewise hazards (last element <code>Inf</code>).
<code>d.hazard</code>	Dropout hazard(s), same structure as <code>e.hazard</code> .
<code>d.median</code>	Dropout median(s); an alternative to <code>d.hazard</code> .
<code>d.time</code>	Dropout breakpoints for piecewise hazards.
<code>seed</code>	Optional integer seed for the <code>dqrng</code> generator.
<code>prevalence</code>	Optional subgroup prevalence specification (numeric vector, list of vectors, array, or a named control/treatment list for group-specific prevalence), for the single-endpoint and the illness-death models.
<code>fixed.alloc</code>	Logical; when <code>TRUE</code> subgroup sizes are deterministic rather than drawn.
<code>h01.hazard</code>	Transition hazard(s) for the non-terminal (intermediate) event (state 0 to state 1) in the illness-death model. A scalar or vector for one group, or a two-element list for two groups. Supplying any of <code>h01.*</code> or <code>h02.*</code> activates the illness-death model with two correlated endpoints, and is mutually exclusive with <code>e.hazard</code> / <code>e.median</code> .
<code>h01.median</code>	Median(s) for the intermediate event; an alternative to <code>h01.hazard</code> .
<code>h01.time</code>	Breakpoints for a piecewise <code>h01.hazard</code> (last element <code>Inf</code>).

<code>h02.hazard</code>	Transition hazard(s) for the terminal event without a prior intermediate event (state 0 to state 2). Same group and piecewise conventions as <code>h01.hazard</code> .
<code>h02.median</code>	Median(s) for the direct terminal event; an alternative to <code>h02.hazard</code> .
<code>h02.time</code>	Breakpoints for a piecewise <code>h02.hazard</code> .
<code>h12.hazard</code>	Transition hazard(s) for the terminal event after an intermediate event (state 1 to state 2) for subjects who do not switch. Defaults to <code>h02.hazard</code> , which gives the Fleischer maximal-independence model (Fleischer Theorem 1 when there is no switching and the hazards are constant; with a piecewise <code>h02.hazard</code> the clock-reset <code>h12</code> restarts the piecewise profile at the intermediate event).
<code>h12.median</code>	Median(s) for the post-event terminal event; an alternative to <code>h12.hazard</code> .
<code>h12.time</code>	Breakpoints for a piecewise <code>h12.hazard</code> , measured from the intermediate-event time (clock-reset).
<code>switch.prop</code>	Probability that a subject whose intermediate event is observed (on or before dropout) switches treatment at that event, a scalar or a two-element list (control, treatment). Defaults to zero (no switching); the treatment group is typically left at zero.
<code>h12.switch.hazard</code>	Transition hazard(s) from state 1 to state 2 for subjects who switch. Required when any <code>switch.prop</code> is positive.
<code>h12.switch.median</code>	Median(s) for the post-switch terminal event; an alternative to <code>h12.switch.hazard</code> .
<code>h12.switch.time</code>	Breakpoints for a piecewise <code>h12.switch.hazard</code> , measured from the switch (intermediate-event) time (clock-reset).
<code>switch.clock</code>	Time origin for the post-event hazards. Currently only "reset" (measured from the intermediate event) is implemented.
<code>stream</code>	Optional non-negative whole number selecting an independent dqrng random-number stream for the given seed; requires seed. A large simulation can be split into batches that are generated with the same seed and <code>stream = 1, 2, ...</code> , sequentially or in parallel, and the result of each batch does not depend on how the batches are distributed. With the default Xoroshiro128++ generator of dqrng, stream <code>k</code> starts <code>k</code> jumps of 2^{64} draws ahead of the seeded state, so <code>stream = 0</code> gives the same numbers as no stream. See the section on batches.

Details

For each subject the observed time-to-event is `tte = pmin(surv_time, dropout_time)` and event is 1 when the survival time occurs first. The calendar time of the observed event is `accrual_time + tte`.

A hazard of zero in the last piece (for example a cure fraction, as in `e.hazard = c(0.1, 0)` with `e.time = c(0, 24, Inf)`) gives an infinite latent time to some subjects. A subject with neither a finite survival time nor a finite dropout time has `tte = Inf` and `event = 0`, and an analysis at a finite cutoff censors the subject at the cutoff.

The total enrolled is fixed at `sum(n)`. With `a.rate` the rates are absolute (subjects per unit time): when the accrual period is fully specified the rates must accrue exactly `sum(n)`, and when one extra

rate is given the end of the final interval is solved so the total is met. With `a.prop` the values are relative proportions that distribute `sum(n)` across the fully specified intervals. Each accrual interval receives a deterministic number of subjects (the rate or proportion times the group total, rounded to keep the per-group total exact), placed uniformly within the interval.

Survival and dropout are exponential when a single hazard (or median) is supplied and piecewise-exponential when a vector is supplied together with the corresponding `e.time` or `d.time` break-points, whose last element must be `Inf`. Group-specific parameters are supplied as a two-element list (control first, treatment second).

When prevalence is supplied the trial has subgroups. A numeric vector defines a single factor; a list of numeric vectors defines several independent factors; a multi-dimensional array defines the joint distribution of correlated factors. Per-cell hazards may be supplied as a list with one element per cell (cells in column-major order, the first factor varying fastest). With `fixed.alloc = TRUE` the subgroup sizes are deterministic and the fixed labels are assigned to subjects in a random order within each simulated trial, so subgroup membership does not depend on the accrual time; otherwise subgroup membership is drawn from the prevalence distribution.

The number of groups is determined as follows. A length-two `n` always gives two groups. A scalar `n` gives two groups, with the total split by `alloc`, in any of these cases: `alloc` is supplied explicitly; the prevalence is group-specific; without subgroups, any of `e.hazard`, `e.median`, `d.hazard`, and `d.median` is a list; with subgroups, one of them is a length-two list with a list element (per-cell values within a group, as in `list(list(0.10, 0.08, 0.06), 0.05)`). Otherwise a scalar `n` gives one group, and with subgroups a list is read as one element per cell of that group. So with subgroups and a scalar `n`, `e.hazard = list(0.10, 0.05)` means per-cell hazards of a single group unless `alloc` is supplied, in which case it means per-group hazards. In a two-group simulation each specification is either shared by both groups (not a list) or a list of length two (control first), and each group's element may be a per-cell list when there are subgroups. Without subgroups, a length-two `n` requires at least one of the survival and dropout specifications to be such a list. The breakpoints `e.time` and `d.time` follow the same structure (per group in a two-group simulation, per cell in a one-group simulation with subgroups).

When `n` is a vector of length greater than two together with a per-arm survival list, the simulation is a multi-arm trial. Each arm is generated in turn with the validated single-group kernel over a common accrual window, and the arms are stacked into one data frame with a group column labeled 1 to `length(n)` in the order of `n`. Per-arm survival is supplied as an `e.hazard` or `e.median` list with one element per arm, and optional dropout as a shared value or a per-arm list through `d.hazard` or `d.median`. The arms share the master seed, so the result is reproducible. A multi-arm design is analyzed as a set of pairwise contrasts by subsetting the output to the control arm and one other arm and calling `analysis_fast` once per contrast, which `pairwise_fast` does for every experimental arm. Multi-arm mode does not support subgroups or the illness-death model, which remain two-group.

The illness-death model (activated by `h01.*` or `h02.*`) also accepts prevalence and `fixed.alloc`. The transition hazards (`h01.*`, `h02.*`, `h12.*`, `h12.switch.*`), `switch.prop`, and the dropout specification then follow the rules of `e.hazard` with subgroups: in a two-group simulation each is shared or a list with one element per group, and each group's element is shared or a list with one element per subgroup cell, for example `h01.hazard = list(list(0.10, 0.06), 0.05)`; with a scalar `n` and no `alloc`, a list without list elements holds one element per cell of a single group. All subjects share one accrual process and the subgroup cells are assigned after accrual, so the subgroups enroll over the same calendar in proportion to their prevalence. This gives a mixture of illness-death models without generating the subgroups separately and adjusting their accrual rates. With a single cell

(for example prevalence = 1) the data equal those without prevalence, with the subgroup column added.

Value

A data.frame with $\text{nsim} * \text{sum}(n)$ rows. The columns are sim, group, any subgroup columns, accrual_time, surv_time, dropout_time, tte, event, and calendar_time. In the illness-death model the columns are instead sim, group, any subgroup columns, accrual_time, e1_surv_time, e2_surv_time, dropout_time, e1_tte, e1_event, e2_tte, e2_event, e1_calendar_time, e2_calendar_time, intermediate, switched, and switch_time, where e1 is the first (state-0 exit) endpoint and e2 is the terminal endpoint. In oncology e1 is progression-free survival, e2 is overall survival, and intermediate flags progression. The column intermediate describes the latent process (an intermediate event before the terminal event, $\text{e1_surv_time} < \text{e2_surv_time}$), also when it would occur after dropout; the observed intermediate event is $\text{intermediate} == 1$ with $\text{e1_event} == 1$. A switch ($\text{switched} == 1$, at switch_time , the intermediate-event time) occurs only after an observed intermediate event.

Batches and parallel execution

The output has $\text{nsim} * \text{sum}(n)$ rows, so memory rather than time limits the number of simulated trials in one call. A large study can be run in batches: each batch is generated with the same seed and its own stream, analyzed with [analysis_fast](#), and only the analysis results are kept. Because each stream is an independent sequence of the dqrng generator, the data of batch b are the same whether the batches are run one after another or distributed over parallel workers (for example with `parallel::mclapply()` or the future framework), and whatever the number of workers. The simulation identifiers start at 1 in every batch, so they are renumbered before the batch results are combined. Functions that draw further random numbers, such as [switch_fast](#), continue the same stream within a batch. The max-combo p-values of [analysis_fast](#) with four or more weights (or a two-sided test) are computed with R's own random-number generator, so a batch should also call `set.seed()` before the analysis for p-values that are identical across runs.

See Also

[analysis_fast](#), [cutoff_fast](#), [switch_fast](#), [pairwise_fast](#)

Examples

```
# Batches from independent random-number streams: the simulations of each
# batch are renumbered before the analysis results are combined.
run_batch <- function(b, nsim = 20) {
  d <- simdata_fast(nsim = nsim, n = c(100, 100), a.time = c(0, 12),
    a.rate = 200 / 12, e.median = list(12, 16),
    seed = 2026, stream = b)
  r <- analysis_fast(d, control = 1, event.looks = 120)
  r$sim <- r$sim + (b - 1) * nsim
  r
}
res <- do.call(rbind, lapply(1:3, run_batch))
range(res$sim)
```

```

# One-group simulation, simple exponential, no dropout
df1 <- simdata_fast(
  nsim      = 100,
  n         = 50,
  a.time    = c(0, 12),
  a.rate    = 50 / 12,
  e.median  = 18,
  seed      = 1
)
head(df1)

# Accrual rate with the final interval computed from the total: 20 per unit
# time for the first 12 units, then 30 per unit time until 500 are enrolled
df1b <- simdata_fast(
  nsim      = 100,
  n         = 500,
  a.time    = c(0, 12),
  a.rate    = c(20, 30),
  e.median  = 18,
  seed      = 1
)
head(df1b)

# Accrual by proportion: 30% enrolled in [0, 6], 70% in [6, 12]
df1c <- simdata_fast(
  nsim      = 100,
  n         = 50,
  a.time    = c(0, 6, 12),
  a.prop    = c(0.3, 0.7),
  e.median  = 18,
  seed      = 1
)
head(df1c)

# Two-group simulation, simple exponential, with dropout
df2 <- simdata_fast(
  nsim      = 100,
  n         = c(60, 60),
  a.time    = c(0, 6, 12),
  a.rate    = c(8, 12),
  e.median  = list(18, 24),
  d.hazard  = list(0.01, 0.01),
  seed      = 2
)
head(df2)

# One factor with three levels: single subgroup column
df3 <- simdata_fast(
  nsim      = 100,
  n         = c(150, 150),
  a.time    = c(0, 12),
  a.rate    = 300 / 12,
  e.hazard  = list(list(0.10, 0.08, 0.06), 0.05),

```

```

    prevalence = c(0.5, 0.3, 0.2),
    seed       = 3
  )
head(df3)

# The same trial specified by the total sample size and the allocation ratio
df3b <- simdata_fast(
  nsim      = 100,
  n         = 300,
  alloc     = c(1, 1),
  a.time    = c(0, 12),
  a.rate    = 300 / 12,
  e.hazard  = list(list(0.10, 0.08, 0.06), 0.05),
  prevalence = c(0.5, 0.3, 0.2),
  seed     = 3
)
identical(df3, df3b)

# Two independent factors (2 x 2): columns subgroup1 and subgroup2.
# Four cells in column-major order: (1,1), (2,1), (1,2), (2,2).
df4 <- simdata_fast(
  nsim      = 100,
  n         = 200,
  a.time    = c(0, 12),
  a.rate    = 200 / 12,
  e.hazard  = list(0.10, 0.08, 0.07, 0.05),
  prevalence = list(c(0.5, 0.5), c(0.6, 0.4)),
  seed     = 4
)
head(df4)

# Two correlated factors via a joint-distribution array (2 x 2)
df5 <- simdata_fast(
  nsim      = 100,
  n         = 200,
  a.time    = c(0, 12),
  a.rate    = 200 / 12,
  e.hazard  = 0.08,
  prevalence = array(c(0.40, 0.10, 0.15, 0.35), dim = c(2, 2)),
  seed     = 5
)
head(df5)

# Two correlated endpoints, no switching (reduces to Fleischer Theorem 1
# because h12 defaults to h02). In oncology e1 is PFS and e2 is OS; here the
# control has faster intermediate events and faster direct terminal events.
dfid <- simdata_fast(
  nsim      = 100,
  n         = c(150, 150),
  a.time    = c(0, 12),
  a.rate    = 300 / 12,
  h01.median = list(8, 12),
  h02.median = list(24, 30),

```

```

    seed      = 6
  )
  head(dfid)

# With treatment switching: 40 percent of control subjects who reach the
# intermediate event switch and then follow a more favorable post-event hazard
dfsw <- simdata_fast(
  nsim      = 100,
  n         = c(150, 150),
  a.time    = c(0, 12),
  a.rate    = 300 / 12,
  h01.median = list(8, 12),
  h02.median = list(24, 30),
  switch.prop = list(0.4, 0),
  h12.switch.median = list(36, 36),
  seed      = 7
)
head(dfsw)

# Illness-death model with two subgroups (prevalence 0.4 and 0.6): the
# treatment effect on progression is larger in subgroup 2.
dfsg <- simdata_fast(
  nsim      = 100,
  n         = c(150, 150),
  a.time    = c(0, 12),
  a.rate    = 300 / 12,
  h01.median = list(8, list(10, 14)),
  h02.median = list(24, 30),
  prevalence = c(0.4, 0.6),
  seed      = 8
)
table(dfsg$group, dfsg$subgroup) / 100

# Three-arm trial (one control and two treatment arms) analyzed as pairwise
# contrasts against the shared control.
dfk <- simdata_fast(
  nsim      = 100,
  n         = c(120, 120, 120),
  a.time    = c(0, 12),
  a.rate    = 360 / 12,
  e.median = list(12, 16, 20),
  seed      = 8
)
# Control arm (group 1) versus treatment arm 2, one-sided log-rank at month 24.
sub12 <- dfk[dfk$group %in% c(1, 2), ]
res12 <- analysis_fast(sub12, control = 1, time.looks = 24, side = 1)
head(res12)

```

simsummary_fast *Summarize Operating Characteristics from Sequential Analysis Output*

Description

Aggregates the per-simulation, per-look output of [analysis_fast](#) into operating characteristics: the rejection rate, the futility-stopping rate, the distribution of the stopping look, and the expected analysis timing (events and calendar time at stopping). A group-sequential design is summarized by applying the supplied per-look boundaries in sequence, and a fixed design is the single-look ($K = 1$) degenerate case of the same logic. This function consumes boundaries computed elsewhere (for example by `gsDesign` or `rpact`) and does not compute or spend α itself; it estimates the stopping probabilities by Monte Carlo over the simulated trials rather than by the analytic numerical integration used by those packages, so the two agree only up to Monte Carlo error and converge as the number of simulations grows.

Usage

```
simsummary_fast(
  data,
  eff.col = NULL,
  efficacy = NULL,
  fut.col = eff.col,
  futility = NULL,
  direction = c("lower", "upper"),
  p.col = NULL,
  alpha = NULL
)
```

Arguments

<code>data</code>	A data frame from analysis_fast , containing at least <code>sim</code> and <code>look</code> , the statistic columns named by the boundary arguments, and, when available, <code>n.event</code> and <code>cutoff</code> for the analysis-timing summaries.
<code>eff.col</code>	A single character naming the statistic column compared with efficacy. Required with the Z mode.
<code>efficacy</code>	A numeric vector of efficacy boundaries, one per look, on the scale of <code>eff.col</code> . Entries may be NA to omit the efficacy test at that look. Required with the Z mode.
<code>fut.col</code>	A single character naming the statistic column compared with futility. Defaults to <code>eff.col</code> . Used only in the Z mode and only when <code>futility</code> is supplied; it may differ from <code>eff.col</code> so that futility is judged on a different scale (for example a log hazard ratio).
<code>futility</code>	A numeric vector of futility boundaries, one per look, on the scale of <code>fut.col</code> , or NULL for no futility stopping. Entries may be NA to omit a futility boundary at that look. Used only in the Z mode.

direction	A single string, either "lower" (default) or "upper", giving the direction in which the boundaries are crossed. See Details.
p.col	A single character naming the p-value column compared with alpha. Selects the p mode; use either the Z mode (eff.col with efficacy) or the p mode (p.col with alpha), not both.
alpha	A numeric vector of per-look nominal significance levels, one per look. Entries may be NA to omit the test at a look. Required with p.col.

Details

The boundaries are supplied per look, one value for each distinct value of the look column in data. Two boundary modes are available and exactly one must be used.

In the Z mode the efficacy boundary is compared with the column named by `eff.col` and the futility boundary with the column named by `fut.col`. The two columns may differ, which lets the efficacy and futility rules live on different scales. For example a beta-spending efficacy boundary can be applied to the standardized statistic `logrank.z` while a futility boundary expressed as a log hazard ratio is applied to `cox.coef`. The crossing direction is set by `direction`: with "lower" (the natural sign of `logrank.z`, `cox.z`, and `cox.coef`, where treatment benefit is negative) the efficacy boundary is crossed when the efficacy statistic is at or below efficacy and the futility boundary is crossed when the futility statistic is at or above futility; with "upper" the inequalities are reversed. Either boundary vector may contain NA at some looks to omit that rule there, so efficacy-only and futility-only looks are expressed by placing NA in the other vector. A look with NA on both rules can never stop the trial.

In the p mode the p-value in the column named by `p.col` is compared with the per-look nominal level alpha, rejecting when the p-value is at or below the level. Futility is not used in the p mode, and alpha may contain NA to omit the efficacy test at a look.

For each simulated trial the looks are examined in order. The trial stops at the first look whose statistic crosses a boundary. Crossing the efficacy boundary is a rejection of the null hypothesis; crossing the futility boundary (Z mode only) is a stop without rejection. When both are crossed at the same look the efficacy stop takes precedence. A look whose relevant statistic is NA, or whose boundary is NA, triggers no crossing of that rule and the trial continues. A trial that reaches the final look without crossing the efficacy boundary does not reject.

Each look's `prob.stop.efficacy` is the marginal probability of stopping for efficacy for the first time at that look, that is the probability of not stopping at any earlier look and crossing the efficacy boundary at this one. This is the stage-wise rejection contribution of a group-sequential design; the cumulative sum `cum.reject` is the cumulative power up to and including that look, and the total over all looks is `rejection.rate`. These are the same quantities that `gsDesign` and `rpact` report, estimated here by simulation.

A simulation whose event target was not reached at a look (`reached = FALSE` in `analysis_fast` output) carries the statistics of its final data and an NA cutoff at that look. Its statistics are compared with the boundary like any other, and it is left out of `cutoff.mean` (the mean over the simulations with a finite cutoff) but not out of the count means. In `pairwise_fast` output the statistics and counts of an unreached look are NA, so that look never crosses a boundary and is left out of the count means.

The rejection rate is the type I error under a null data-generating truth and the power under an alternative truth, but because the function does not know the truth used to generate data it is reported neutrally as the rejection rate and its interpretation is left to the user.

When data carries a population column (the long form produced by [analysis_fast](#) with `by.subgroup = TRUE`), the same boundaries are applied within each population and the output has one block of rows per population. Likewise, when data carries an arm column (the output of [pairwise_fast](#)), each experimental arm's contrast against the control is summarized separately, and the output gains an arm column with one block of rows per arm (and population). Within a block, data must have at most one row per simulation and look.

Value

An object of class "simsummary_fast": a data frame with one row per population and look plus an overall summary row appended after each population's looks. The columns are arm (only when data has an arm column), population, look (the look index, or "overall" on the summary row), optionally look.value, n.enrolled.mean and n.event.mean (the mean enrolled and event counts at that look, or at the stopping look on the summary row), n.dropout.mean and n.pipeline.mean (the mean dropout count and pipeline count $n.enrolled - n.event - n.dropout$, when those columns are present in data), cutoff.mean (the mean calendar time, likewise), prob.stop.efficacy, prob.stop.futility, prob.stop.any, and cum.reject. On the overall row prob.stop.efficacy is the total rejection rate, prob.stop.futility the total futility rate, prob.stop.any their sum, and cum.reject again the total rejection rate; its timing columns are the expected counts and calendar time at the stopping look. The number of simulations is stored in the attribute nsim (one value per block when the blocks differ) and the boundary settings in the attribute boundary.

See Also

[analysis_fast](#), [simdata_fast](#), [print.simsummary_fast](#).

Examples

```
df <- simdata_fast(
  nsim    = 200,
  n       = c(150, 150),
  a.time  = c(0, 12),
  a.rate  = 300 / 12,
  e.hazard = list(0.05, 0.035),
  seed    = 1
)

res <- analysis_fast(df, control = 1, event.looks = c(60, 105, 150),
  stat = c("logrank", "coxph"), side = 1)

# Efficacy on the standardized log-rank Z, futility on the log hazard ratio,
# with a futility-only first look and efficacy-only later looks
simsummary_fast(res,
  eff.col = "logrank.z",
  efficacy = c(NA, -2.96, -1.97),
  fut.col = "cox.coef",
  futility = c(log(1.2), NA, NA),
  direction = "lower")

# p-value boundaries instead
simsummary_fast(res, p.col = "logrank.p",
```

```
alpha = c(0.0006, 0.0151, 0.0245))
```

survdiff_fast

Fast Log-Rank Test for Two-Group Survival Data

Description

Computes the log-rank test statistic for comparing survival curves between two groups. Returns either a one-sided Z-score or a two-sided chi-square statistic. The C++ backend uses a two-pointer merge scan over pooled sorted vectors, eliminating the `rank()` call that dominates the pure-R implementation. When a `strata` argument is supplied, the stratified log-rank test is computed instead, matching `survdiff` with a `strata()` term. A non-default `weight` argument selects a weighted log-rank test (Fleming-Harrington, modestly-weighted, Gehan-Breslow, or Tarone-Ware) for non-proportional hazards.

Usage

```
survdiff_fast(
  time,
  event,
  group,
  control,
  side = 2,
  presorted = FALSE,
  strata = NULL,
  weight = c("logrank", "fh", "mwlrt", "gehan", "tarone-ware"),
  rho = 0,
  gamma = 0,
  t_star = NULL
)
```

Arguments

<code>time</code>	A numeric vector of follow-up times for all subjects.
<code>event</code>	An integer or numeric vector of event indicators (1 = event, 0 = censored), aligned with <code>time</code> .
<code>group</code>	A vector of group labels aligned with <code>time</code> .
<code>control</code>	A scalar value indicating which level of group represents the control group.
<code>side</code>	An integer, either 1 or 2 (default 2). If <code>side = 1</code> , returns the standardized log-rank statistic (Z-score), defined as $(O_1 - E_1) / \sqrt{V_1}$ for the treatment group, so the Z-score is negative when the treatment group has fewer events than expected (a protective treatment effect). If <code>side = 2</code> , returns the chi-square statistic (Z^2).

presorted	A logical value. If TRUE, time, event, and group (and strata when supplied) are assumed to be already sorted in the required order, and the internal order() call is skipped. If FALSE (default), sorting is handled internally. See Details for the required order in the stratified case. The order is checked, and an error is given when it does not hold.
strata	An optional vector of stratum labels aligned with time. If NULL (default), the ordinary log-rank test is computed and the behavior is identical to earlier versions of this function. If supplied, the stratified log-rank test is computed, matching survdiff with a strata() term. Any type that supports equality comparison is accepted. May be combined with a non-default weight to obtain a stratified weighted log-rank test.
weight	A character string naming the weight scheme. "logrank" (default) is the ordinary unweighted log-rank test and reproduces the behavior of earlier versions of this function exactly. "fh" is the Fleming-Harrington G(rho, gamma) test with weight $S(t-)^{\rho} (1 - S(t-))^{\gamma}$. "mwlrt" is the modestly-weighted log-rank test of Magirr and Burman with weight $1 / \max(S(t-), S(t_{\text{star}}-))$. "gehan" is the Gehan-Breslow test with weight equal to the at-risk count, and "tarone-ware" uses the square root of the at-risk count. Here $S(t-)$ is the left-continuous pooled Kaplan-Meier estimate just prior to each event time.
rho	A numeric Fleming-Harrington first parameter, used only when weight = "fh". Defaults to 0.
gamma	A numeric Fleming-Harrington second parameter, used only when weight = "fh". Defaults to 0. The pair rho = 0, gamma = 0 reproduces the ordinary log-rank test, and rho = 0, gamma = 1 is the Fleming-Harrington G(0, 1) test for delayed effects.
t_star	A single non-negative numeric value, the timepoint of the modestly-weighted log-rank test. Required only when weight = "mwlrt". The weight is capped at $1 / S(t_{\text{star}}-)$, where $S(t_{\text{star}}-)$ is the pooled Kaplan-Meier value just before t_star (the product over event times strictly less than t_star, as in <code>nphRCT</code>). A value of 0 yields the ordinary log-rank test.

Details

The log-rank statistic is computed as:

$$Z = (O_1 - E_1) / \sqrt{V_1}$$

where O_1 is the observed number of events in the treatment group, E_1 is the expected number under the null hypothesis of equal survival, and V_1 is the hypergeometric variance. Tied event times are handled correctly: all subjects sharing the same event time form a tied block, and the block is processed atomically in the two-pointer merge.

When strata is NULL (default), the ordinary two-group log-rank test is computed by the C++ core `logrank_core`, which walks the pooled sorted data with a single two-pointer scan, maintaining running at-risk counts per group. No rank vector is constructed, so the dominant $O(n \log n)$ cost of `rank()` in the pure-R version is removed.

When strata is supplied, the stratified log-rank test is computed by the C++ core `stratified_logrank_core`. The contributions O_1 , E_1 , and V_1 are accumulated within each stratum and then summed across strata, so the overall statistic is $Z = (\text{sum } O_1 - \text{sum } E_1) / \sqrt{\text{sum } V_1}$. A stratum that contains

only one group contributes zero to all three totals, the same convention used by `survdiff`. This is the standard stratified log-rank test, equivalent to a log-rank test that conditions on the stratum at each event time.

When a non-default weight is requested, the weighted log-rank test is computed by the C++ core `weighted_logrank_core`. The statistic is $Z = U / \sqrt{V}$ with $U = \sum w_j (O_{1j} - E_{1j})$ and $V = \sum w_j^2 n_{0j} n_{1j} O_{0j} (n_{0j} - O_{0j}) / (n_{0j}^2 (n_{0j} - 1))$, where the weight w_j is one of the schemes named by `weight`. The Fleming-Harrington and modestly-weighted schemes use the left-continuous pooled Kaplan-Meier estimate $S(t-)$, initialized at 1 and updated after each event time, so the first event always has $S(t-) = 1$. The modestly-weighted scheme determines its weight cap in a first pass over the event times before accumulating U and V in a second pass; the other schemes accumulate in a single pass. When both `weight` and `strata` are supplied, the stratified weighted log-rank test is computed by the C++ core `stratified_weighted_logrank_core`: each stratum is an independent weighted log-rank test whose weights come from that stratum's own pooled Kaplan-Meier estimate, and the per-stratum U and V are summed before standardizing once as $Z = \sum U / \sqrt{\sum V}$.

When `presorted = TRUE`, the input vectors are assumed to be sorted and the internal `order()` call is skipped. In the unstratified case the assumed order is ascending time. In the stratified case the assumed order is by stratum first and by ascending time within each stratum, so that rows of the same stratum are contiguous. When `presorted = FALSE` (default), sorting is handled internally. In simulation loops where the data are generated in the required order, setting `presorted = TRUE` avoids one $O(n \log n)$ pass.

The returned object has class `"survdiff_fast"` and is a length-one numeric value (Z-score or chi-square) with the underlying counts `O_0`, `E_0`, `O_1`, `E_1`, `V_1`, the requested side, and the total sample size stored as attributes. When `strata` is supplied, the number of strata is also stored as the attribute `strata`. A `print()` method formats the result similarly to `print(survival::survdiff(...))`, displaying observed and expected event counts for both the control and treatment groups.

Value

An object of class `"survdiff_fast"`, which is a length-one numeric value with attributes `O0`, `E0`, `O1`, `E1`, `V1`, `side`, and `n`, plus `strata` (the number of strata) when `strata` is supplied, or `weight` (the scheme name) when a non-default weight is used. The numeric value is the Z-score when `side = 1`, or the chi-square statistic when `side = 2`. For a weighted test the value is $U / \sqrt{V}$ (or its square), `V1` holds the weighted variance V , `O0` and `O1` hold the raw observed event counts, and `E0` and `E1` are `NA` because a single unweighted expected count is not defined for a weighted test. Returns `NA_real_` (still with class `"survdiff_fast"`) when the variance is zero (e.g., all events in one group).

References

- Gehan, E. A. (1965). A generalized Wilcoxon test for comparing arbitrarily singly-censored samples. *Biometrika*, 52, 203-223.
- Mantel, N. (1966). Evaluation of survival data and two new rank order statistics arising in its consideration. *Cancer Chemotherapy Reports*, 50(3), 163-170.
- Tarone, R. E., & Ware, J. (1977). On distribution-free tests for equality of survival distributions. *Biometrika*, 64, 156-160.

Fleming, T. R., & Harrington, D. P. (1991). *Counting Processes and Survival Analysis*. New York: John Wiley & Sons.

Magirr, D., & Burman, C.-F. (2019). Modestly weighted logrank tests. *Statistics in Medicine*, 38(20), 3782-3790.

See Also

[survdiff](#) for the standard implementation. [print.survdiff_fast](#) for the print method.

Examples

```
library(survival)

# Two-sided test: compare with survdiff
fit <- survdiff_fast(ovarian$futime, ovarian$fustat, ovarian$rx, 2, side = 2)
fit

chisq_ref <- survdiff(Surv(futime, fustat) ~ rx, data = ovarian)$chisq
cat("survdiff_fast chi-square:", as.numeric(fit), "\n")
cat("survdiff      chi-square:", chisq_ref,      "\n")

# One-sided test (Z-score)
survdiff_fast(ovarian$futime, ovarian$fustat, ovarian$rx, 2, side = 1)

# presorted = TRUE: sort once outside, reuse inside a loop
ord <- order(ovarian$futime)
survdiff_fast(ovarian$futime[ord], ovarian$fustat[ord], ovarian$rx[ord],
              control = 2, side = 2, presorted = TRUE)

# Stratified log-rank test: compare with survdiff + strata()
fit_str <- survdiff_fast(ovarian$futime, ovarian$fustat, ovarian$rx, 2,
                        side = 2, strata = ovarian$resid.ds)
chisq_str <- survdiff(Surv(futime, fustat) ~ rx + strata(resid.ds),
                     data = ovarian)$chisq
cat("stratified survdiff_fast:", as.numeric(fit_str), "\n")
cat("stratified survdiff      :", chisq_str,      "\n")

# Weighted log-rank tests for non-proportional hazards
# Fleming-Harrington G(0, 1), emphasizing late differences
survdiff_fast(ovarian$futime, ovarian$fustat, ovarian$rx, 2, side = 1,
              weight = "fh", rho = 0, gamma = 1)

# Modestly-weighted log-rank test with t_star = 365
survdiff_fast(ovarian$futime, ovarian$fustat, ovarian$rx, 2, side = 1,
              weight = "mwlrt", t_star = 365)

# Stratified weighted log-rank test: Fleming-Harrington G(0,1) within strata
survdiff_fast(ovarian$futime, ovarian$fustat, ovarian$rx, 2, side = 1,
              weight = "fh", rho = 0, gamma = 1, strata = ovarian$resid.ds)

# Speed comparison against survdiff()
```

```

if (requireNamespace("microbenchmark", quietly = TRUE)) {
  microbenchmark::microbenchmark(
    survdiff_fast = survdiff_fast(ovarian$futime, ovarian$fustat,
                                  ovarian$rx, 2, side = 2),
    survdiff      = survdiff(Surv(futime, fustat) ~ rx, data = ovarian),
    times = 1000
  )
}

```

survfit_fast

*Fast Kaplan-Meier Survival Probability at a Specified Time Point***Description**

Computes the Kaplan-Meier survival probability at a specified time point, together with a standard error and confidence interval based on Greenwood's variance formula. The C++ backend performs binary search for the evaluation cutoff and accumulates the Kaplan-Meier product and Greenwood sum in a single scan over event positions only, without constructing intermediate vectors.

Usage

```

survfit_fast(
  t_sorted,
  e_sorted,
  t_eval,
  conf.level = 0.95,
  conf.type = "log",
  presorted = TRUE
)

```

Arguments

<code>t_sorted</code>	A numeric vector of event or censoring times. Must be sorted in ascending order when <code>presorted = TRUE</code> .
<code>e_sorted</code>	An integer or numeric vector of event indicators (1 = event, 0 = censored), aligned with <code>t_sorted</code> .
<code>t_eval</code>	A single numeric value specifying the time point at which the survival probability is evaluated.
<code>conf.level</code>	A single numeric value in (0, 1) specifying the confidence level. Defaults to 0.95.
<code>conf.type</code>	A character string specifying the confidence interval type. Must be one of "plain", "log", or "log-log". Defaults to "log".
<code>presorted</code>	A logical value. If <code>TRUE</code> (default), <code>t_sorted</code> and <code>e_sorted</code> are assumed to be sorted in ascending order of time. If <code>FALSE</code> , the vectors are sorted internally before computation.

Details

The Kaplan-Meier estimate at time `t_eval` is defined as the product-limit estimator evaluated at the largest observed event time less than or equal to `t_eval`. If `t_eval` is smaller than the first observed event time, $S(t) = 1$ and the standard error is zero.

The standard error is estimated by Greenwood's formula:

$$SE[S(t)] = S(t) * \sqrt{\sum_{t_i \leq t, d_i > 0} d_i / (n_i * (n_i - d_i))}$$

where `d_i` is the number of events and `n_i` is the number at risk at time `t_i`. The output field `std.err` is $SE[S(t)]$, the standard error on the survival scale, which is the `std.err` reported by `summary()` of a `survfit` object. The `std.err` component of the `survfit` object itself is on the cumulative-hazard scale, $SE[S(t)] / S(t)$, whatever `conf.type` is.

When $S(t_{eval}) = 0$ (all subjects have experienced the event by `t_eval`), the standard error is zero and the confidence interval collapses to $[0, 0]$, consistent with `survfit`.

When `presorted = TRUE` (default), `t_sorted` and `e_sorted` are expected to be sorted in ascending order of time; the order is checked (a linear-time test) and an error is raised for unsorted input. When `presorted = FALSE`, the vectors are sorted internally before computation. When the standard error is zero (for example $S(t) = 1$ before the first event), the confidence interval collapses to the estimate.

Three confidence interval types are supported via `conf.type`:

- "plain": Linear interval on the survival scale, $S(t) \pm z * SE$. The bounds are clipped to $[0, 1]$.
- "log": Interval on the log scale (default in `survfit`), $S(t) * \exp(\pm z * SE / S(t))$. The upper bound is capped at 1, as in `survfit`.
- "log-log": Interval on the complementary log-log scale, $S(t)^{\exp(\pm z * SE / (S(t) * \log(S(t))))}$.

The returned object has class `"survfit_fast"` and is a named numeric vector of length 4 with the evaluation time `t_eval`, the confidence level `conf.level`, and the confidence interval type `conf.type` stored as attributes. A `print()` method formats the result similarly to `print(summary(survival::survfit(...`

Value

An object of class `"survfit_fast"`, which is a named numeric vector of length 4 with elements `surv`, `std.err`, `lower`, and `upper`, representing the Kaplan-Meier survival estimate, the Greenwood standard error $SE[S(t)]$, and the lower and upper confidence limits at `t_eval`. The evaluation time, confidence level, and confidence interval type are stored as attributes `t_eval`, `conf.level`, and `conf.type`. Returns a vector of `NA_real_` values (still with class `"survfit_fast"`) when `n` is zero.

References

Kaplan, E. L., & Meier, P. (1958). Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53(282), 457-481.

See Also

`survfit` for the standard Kaplan-Meier estimator. `print.survfit_fast` for the print method.

Examples

```
set.seed(42)
t_raw <- rexp(100, rate = 1 / 10)
e_raw <- rbinom(100, 1, 0.7)

# presorted = TRUE (default): sort once outside, reuse inside a loop
ord <- order(t_raw)
t_s <- t_raw[ord]
e_s <- e_raw[ord]
survfit_fast(t_s, e_s, t_eval = 10, conf.type = "plain")
survfit_fast(t_s, e_s, t_eval = 10, conf.type = "log")
survfit_fast(t_s, e_s, t_eval = 10, conf.type = "log-log")

# presorted = FALSE: sort internally, convenient for one-off calls
survfit_fast(t_raw, e_raw, t_eval = 10, presorted = FALSE)

# Validation against survival::survfit
library(survival)
fit <- survfit(Surv(t_raw, e_raw) ~ 1, conf.type = "plain")
summary(fit, times = 10)
```

switch_fast

Fast Treatment Switching in Simulated Trial Data

Description

Applies a treatment-switching rule to simulated trial data, such as the output of `simdata_fast`, by changing only the outcomes that occur after each subject's switch. Subjects of the selected groups switch either at their intermediate event (for example disease progression), at an opening time after an interim analysis (crossover at a milestone), or at the later of the two, and the remaining time to the terminal event after the switch is either multiplied by an acceleration factor or redrawn from a new (piecewise) exponential hazard. All simulated trials are processed at once with vectorized operations. Because the outcomes observed before the switch are kept exactly, an analysis at or before the opening time gives the same result before and after switching, so a crossover that depends on an interim result can be simulated by analyzing the interim, deciding per simulated trial, and switching afterward.

Usage

```
switch_fast(
  data,
  group,
  prob = 1,
  when = c("intermediate", "cutoff", "later"),
  cutoff = NULL,
  delay = 0,
  sims = NULL,
```

```

    aft.factor = NULL,
    hazard = NULL,
    median = NULL,
    time = NULL,
    seed = NULL,
    stream = NULL
  )

```

Arguments

data	A data frame from simdata_fast (single-endpoint, multi-arm, or illness-death), containing at least <code>sim</code> , <code>group</code> , <code>accrual_time</code> , and the latent columns described in Details.
group	A vector of group labels whose subjects may switch, for example the control group 1.
prob	A single probability in [0, 1] that an eligible subject switches. Defaults to 1.
when	A character string, one of "intermediate", "cutoff", and "later", giving the switch time (see Details).
cutoff	Per-simulation calendar times at which switching opens, used with <code>when = "cutoff"</code> or "later". Either a numeric vector with one element per simulated trial or a one-column matrix from cutoff_fast . As in the <code>cutoff.looks</code> argument of analysis_fast , the names of a vector (or the row names of a matrix) are matched to the values of <code>data\$sim</code> ; without names the elements are taken in the order of the sorted distinct values of <code>data\$sim</code> . NA disables switching in that simulated trial.
delay	A single non-negative number added to <code>cutoff</code> , for example the time needed to implement the crossover after an interim analysis. Defaults to 0.
sims	An optional logical vector with one element per simulated trial (in the order of the sorted distinct values of <code>data\$sim</code>). Switching is applied only in the simulated trials with TRUE. Requires <code>when = "cutoff"</code> or "later".
aft.factor	A single positive acceleration factor applied to the remaining time after the switch. Exactly one of <code>aft.factor</code> , <code>hazard</code> , and <code>median</code> must be supplied.
hazard	Post-switch hazard, a single value or a vector for a piecewise exponential distribution with breakpoints <code>time</code> , measured from the switch.
median	A single post-switch median; an alternative to a single hazard.
time	Breakpoints for a piecewise hazard, starting at 0 and ending with Inf, measured from the switch.
seed	Optional integer seed for the dqrng generator.
stream	Optional non-negative whole number selecting a dqrng stream; requires <code>seed</code> . See simdata_fast .

Details

The function works on the latent columns of [simdata_fast](#). For a single-endpoint simulation these are `surv_time` and `dropout_time`; for an illness-death simulation they are `e1_surv_time` (time to

the first event, such as progression-free survival), e2_surv_time (time to the terminal event, such as overall survival), dropout_time, and intermediate. The observed columns (tte, event, and calendar_time, or their e1_ and e2_ counterparts) are recomputed from the modified latent times, and the columns switched and switch_time (time from accrual to the switch) record the switches. The columns sim, group, accrual_time, and the subgroup columns are unchanged.

The switch time s , measured from accrual, is

- when = "intermediate": the intermediate event time e1_surv_time of a subject whose intermediate event occurred (intermediate == 1); illness-death data only;
- when = "cutoff": the opening time cutoff + delay of the subject's simulated trial minus the accrual time (zero for a subject accrued after the opening);
- when = "later": the later of the intermediate event time and the opening time, for a subject whose intermediate event occurred; illness-death data only.

A subject of the selected group switches with probability prob when the switch occurs before the terminal event and before dropout ($s < e2_surv_time$, or $s < surv_time$ for single-endpoint data, and $s < dropout_time$) and the subject has not already switched (switched == 1, for example at progression through the switch.prop argument of [simdata_fast](#)). With when = "cutoff" or "later", the switch never occurs before the opening time, so every outcome observed by calendar time cutoff + delay is left unchanged; subjects accrued after the opening are also eligible, as in a protocol that opens crossover from that time on.

The effect of switching on the terminal time $T > s$ is

- aft.factor = f: $T_{new} = s + f * (T - s)$, the causal accelerated failure time model of the rank-preserving structural failure time (RPSFT) method, under which $f > 1$ prolongs the remaining survival;
- hazard (or median): $T_{new} = s + R$, where the remaining time R is drawn from the (piecewise) exponential distribution with hazard hazard and breakpoints time measured from the switch.

In illness-death data with when = "cutoff", a switch can occur before the intermediate event. The acceleration factor is then applied in the same way to the first-event time when it lies after the switch, which keeps the first event no later than the terminal event; a redrawn hazard is not available in this case because the latent time to the intermediate event after the terminal event is not stored.

sim restricts the switching to some simulated trials, for example those in which the interim analysis crossed an efficacy boundary. It is available with when = "cutoff" or "later" only, so that the decision cannot change outcomes observed before it is made.

The random numbers (one uniform per row when prob < 1, and one exponential per row when hazard or median is used) are drawn with dqrng, in row order, so the result is reproducible from seed and stream, or from the state left by a seeded [simdata_fast](#) call.

Value

The data frame data with the latent and observed columns of the switched subjects updated, and the columns switched (0 or 1) and switch_time (time from accrual to the switch, NA for subjects who did not switch) added or updated.

References

Robins, J. M. and Tsiatis, A. A. (1991). Correcting for non-compliance in randomized trials using rank preserving structural failure time models. *Communications in Statistics - Theory and Methods*, 20(8), 2609-2631.

See Also

[simdata_fast](#), [cutoff_fast](#), [analysis_fast](#).

Examples

```
# Crossover at an interim analysis: control subjects still on study at the
# 150th event switch to the experimental treatment, which multiplies their
# remaining survival time by 1.5.
df <- simdata_fast(nsim = 50, n = c(150, 150), a.time = c(0, 12),
                  a.rate = 300 / 12, e.median = list(12, 18),
                  d.hazard = 0.01, seed = 1)
ia <- cutoff_fast(df, event.looks = 150)
dfs <- switch_fast(df, group = 1, when = "cutoff", cutoff = ia,
                  aft.factor = 1.5)
mean(dfs$switched[df$group == 1])

# The interim analysis is unchanged; the final analysis is diluted.
r0 <- analysis_fast(df, control = 1, cutoff.looks = ia)
r1 <- analysis_fast(dfs, control = 1, cutoff.looks = ia)
all.equal(r0$logrank.z, r1$logrank.z)
fa0 <- analysis_fast(df, control = 1, event.looks = 220)
fa1 <- analysis_fast(dfs, control = 1, event.looks = 220)
c(mean(fa0$logrank.z), mean(fa1$logrank.z))

# Switching at progression in an illness-death simulation: 50 percent of
# the control subjects who progress switch, with a post-progression median
# of 15 instead of 10.
id <- simdata_fast(nsim = 50, n = c(150, 150), a.time = c(0, 12),
                  a.rate = 300 / 12, h01.median = list(6, 9),
                  h02.median = list(20, 24), h12.median = list(10, 10),
                  seed = 2)
ids <- switch_fast(id, group = 1, prob = 0.5, when = "intermediate",
                  median = 15)
table(ids$group, ids$switched)
```

wkm_fast

Fast weighted Kaplan-Meier (Pepe-Fleming) two-sample test

Description

Computes the weighted Kaplan-Meier (WKM) statistic of Pepe and Fleming for comparing two survival curves. The statistic is the weighted integral of the difference between the Kaplan-Meier

estimates of the treatment and control groups over the observed range, standardized to a Wald z statistic. Unlike weighted log-rank tests, this test targets the integrated difference in survival and is sensitive to differences even when the hazard functions cross. The single scan over the sorted data is performed in C++ for use inside simulation loops.

Usage

```
wkm_fast(
  time,
  event,
  group,
  control = NULL,
  side = 2,
  conf.level = 0.95,
  weight = c("PF", "sqrtPF", "constant"),
  presorted = FALSE
)
```

Arguments

time	Numeric vector of event or censoring times.
event	Integer vector, 1 for an event and 0 for censoring.
group	Grouping vector with exactly two distinct levels.
control	The level of group that denotes the control group.
side	Either 2 for a two-sided test or 1 for a one-sided test of treatment superiority (weighted difference greater than 0).
conf.level	Confidence level for the interval of the weighted difference.
weight	Weight function, one of "PF" (Pepe-Fleming combined censoring weight), "sqrtPF" (its square root) or "constant" (weight 1).
presorted	Logical; set to TRUE when time, event and group are already sorted in ascending order of time, to skip the internal sort. Defaults to FALSE. The order is checked, and an error is given when it does not hold.

Details

The default weight is the Pepe-Fleming combined censoring weight $w(t) = n G_1(t) G_2(t) / (n_1 G_1(t) + n_2 G_2(t))$, where G_1 and G_2 are the Kaplan-Meier estimates of the censoring survival functions of the two groups. This weight gives less weight to regions with heavy censoring and stabilizes the variance in the tail. The choice `weight = "sqrtPF"` uses its square root, and `weight = "constant"` uses a weight of 1, in which case the numerator reduces to the difference in restricted mean survival time over the observed range. With `weight = "PF"` the result reproduces `nphsim::wkm.Stat` for data without tied times.

The weighted difference is computed as treatment minus control, so a positive value and a positive z indicate longer survival under treatment.

Value

A named numeric vector of class "wkm_fast" with the weighted integrated difference, its standard error and confidence limits, and the Wald statistics for the test.

References

Pepe, M. S., & Fleming, T. R. (1989). Weighted Kaplan-Meier statistics: a class of distance tests for censored survival data. *Biometrics*, 45(2), 497-507.

Pepe, M. S., & Fleming, T. R. (1991). Weighted Kaplan-Meier statistics: large sample and optimality considerations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 53(2), 341-352.

Examples

```
set.seed(1)
n <- 300
g <- rep(0:1, each = n / 2)
tt <- c(rexp(n / 2, log(2) / 12), rexp(n / 2, log(2) / 16))
cc <- rexp(n, rate = 0.02)
time <- pmin(tt, cc)
event <- as.integer(tt <= cc)
wkm_fast(time, event, group = g, control = 0)
```

wmst_fast

Fast window mean survival time and between-group difference

Description

Estimates the window mean survival time (WMST) for a single group, or the difference in WMST between a treatment group and a control group, together with standard errors, confidence intervals and a Wald test for the difference. The WMST is the area under the Kaplan-Meier curve between the window limits tau1 and tau2. With tau1 = 0 it is the restricted mean survival time at tau2, and the result then matches survRM2::rmst2. The single scan over the sorted data is performed in C++ for use inside simulation loops.

Usage

```
wkst_fast(
  time,
  event,
  group = NULL,
  control = NULL,
  tau1 = 0,
  tau2 = NULL,
  side = 2,
  conf.level = 0.95,
  presorted = FALSE
)
```

Arguments

time	Numeric vector of event or censoring times.
event	Integer vector, 1 for an event and 0 for censoring.
group	Optional grouping vector with exactly two distinct levels for a two-group comparison. If omitted, a single-group WMST is returned.
control	The level of group that denotes the control group. Required when group is supplied.
tau1	Lower limit of the window. Defaults to 0.
tau2	Upper limit of the window. If omitted, the largest time common to both groups is used (the minimum over groups of the maximum observed time), or the maximum observed time for a single group. A supplied value larger than that gives a warning, because the Kaplan-Meier curve is not estimated beyond it.
side	Either 2 for a two-sided test or 1 for a one-sided test of treatment superiority (difference greater than 0).
conf.level	Confidence level for the intervals.
presorted	Logical; set to TRUE when time, event and group are already sorted in ascending order of time, to skip the internal sort. Defaults to FALSE. The order is checked, and an error is given when it does not hold.

Details

The variance of the WMST estimate uses the counting process (Greenwood) form in which each event time at or before tau2 contributes the squared remaining window area. Event times at or before tau1 contribute the full window area, since the hazard before tau1 scales the survival curve throughout the window.

The difference is computed as treatment minus control, so a positive difference indicates a longer mean survival time within the window under treatment.

Value

A named numeric vector of class "wmst_fast". For a single group the elements are the WMST, its standard error and confidence limits. For two groups the elements are the control and treatment WMST values, their difference, the standard errors, the confidence limits and the Wald statistics for the difference.

References

Paukner, M., & Chappell, R. (2021). Window mean survival time. *Statistics in Medicine*, 40(25), 5521-5533.

Examples

```
set.seed(1)
n <- 200
g <- rep(0:1, each = n / 2)
tt <- c(rexp(n / 2, 0.1), rexp(n / 2, 0.07))
```

```
cc <- rexp(n, rate = 0.02)
time <- pmin(tt, cc)
event <- as.integer(tt <= cc)
wmst_fast(time, event, group = g, control = 0, tau1 = 2, tau2 = 12)
```

Index

ahr_fast, [3](#), [7](#), [12](#), [13](#), [22](#), [23](#), [34](#)
ahsw_fast, [4](#), [5](#), [5](#), [7](#), [12](#), [13](#), [37](#)
analysis_fast, [7](#), [18–20](#), [30–32](#), [57](#), [58](#),
[62–64](#), [72](#), [74](#)

coxph, [17](#)
coxph_fast, [5](#), [7](#), [11](#), [13](#), [14](#), [22](#), [23](#), [34](#), [38](#)
cutoff_fast, [7](#), [10](#), [11](#), [13](#), [17](#), [31](#), [32](#), [58](#), [72](#),
[74](#)

format, [38](#)

gen_scenario_fast, [20](#), [35](#), [36](#), [44](#)
GenzBretz, [25](#)

kmcurve_fast, [22](#), [33](#), [34](#), [39](#)

legend, [34](#), [36](#)

maxcombo_fast, [7](#), [11](#), [13](#), [23](#), [39](#)
medsurv_fast, [7](#), [10](#), [12](#), [13](#), [26](#), [40](#)
milestone_fast, [7](#), [9](#), [12](#), [13](#), [28](#), [41](#)

pairwise_fast, [10](#), [11](#), [13](#), [18](#), [20](#), [30](#), [45](#), [57](#),
[58](#), [63](#), [64](#)
plot.kmcurve_fast, [22](#), [23](#), [32](#), [39](#)
plot.scenario_fast, [20](#), [21](#), [35](#), [44](#)
print.ahr_fast, [36](#)
print.ahsw_fast, [37](#)
print.coxph_fast, [17](#), [38](#)
print.kmcurve_fast, [22](#), [23](#), [34](#), [38](#)
print.maxcombo_fast, [39](#)
print.medsurv_fast, [40](#)
print.milestone_fast, [41](#)
print.rmst_fast, [42](#)
print.rmw_fast, [43](#), [53](#)
print.scenario_fast, [20](#), [21](#), [36](#), [43](#)
print.simsummary_fast, [44](#), [64](#)
print.survdiff_fast, [45](#), [68](#)
print.survfit_fast, [46](#), [70](#)
print.wkm_fast, [47](#)

print.wmst_fast, [48](#)

rmst_fast, [7](#), [13](#), [22](#), [23](#), [34](#), [42](#), [49](#)
rmw_fast, [7](#), [9](#), [12](#), [13](#), [43](#), [51](#)

simdata_fast, [8](#), [10](#), [13](#), [18](#), [20–22](#), [30](#), [32](#),
[54](#), [64](#), [71–74](#)
simsummary_fast, [12](#), [31](#), [44](#), [45](#), [61](#)
survdiff, [65–68](#)
survdiff_fast, [7](#), [9](#), [13](#), [24](#), [25](#), [39](#), [45](#), [46](#),
[53](#), [65](#)
survfit, [50](#), [70](#)
survfit_fast, [7](#), [13](#), [22](#), [23](#), [34](#), [46](#), [50](#), [69](#)
switch_fast, [13](#), [18](#), [20](#), [58](#), [71](#)

TVPACK, [25](#)

wkm_fast, [7](#), [10](#), [12](#), [13](#), [47](#), [74](#)
wmst_fast, [8](#), [12](#), [13](#), [48](#), [76](#)