Package 'TemporalHazard'

November 8, 2026

Title Temporal Parametric Hazard Modeling

Version 1.3.0

URL https://ehrlinger.github.io/TemporalHazard/,
https://github.com/ehrlinger/TemporalHazard

BugReports https://github.com/ehrlinger/TemporalHazard/issues

Description Provides native R implementations of the multiphase parametric hazard model of Blackstone, Naftel, and Turner (1986) <doi:10.1080/01621459.1986.10478314> with a focus on behavioral parity, transparent numerics, and reproducible validation against reference outputs from the original 'C'/'SAS' 'HAZARD' program, originally developed at the University of Alabama at Birmingham (UAB). The 'SAS'/'C' code and this R package are currently developed and maintained at The Cleveland Clinic Foundation, and the R code was wholly developed at The Cleveland Clinic Foundation. The generalized temporal decomposition family extends to longitudinal mixed-effects settings (Rajeswaran et al. 2018 <doi:10.1177/0962280215623583>). The package is intentionally implemented in pure R first; performance-critical paths may later be accelerated with 'Rcpp' without changing the public interface.

License GPL (>= 2)

Copyright See the file COPYRIGHTS in the installed package (inst/COPYRIGHTS in the source) for .hzr_mt_normal().

Encoding UTF-8

Language en-US

VignetteBuilder quarto

SystemRequirements Quarto command line tool (<https://github.com/quarto-dev/quarto-cli>)

Depends R (>= 4.1.0)

Imports survival

Suggests covr, ggplot2, haven, knitr, lintr, numDeriv, pkgdown, quarto, roxygen2, rmarkdown, scales, testthat (>= 3.2.0), withr

LazyData true

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author John Ehrlinger [aut, cre, cph],
R Core Team [cph] (seeding and inversion in .hzr_mt_normal()),
Makoto Matsumoto [cph] (Mersenne Twister in .hzr_mt_normal()),
Takuji Nishimura [cph] (Mersenne Twister in .hzr_mt_normal())

Maintainer John Ehrlinger <john.ehrlinger@gmail.com>

Repository CRAN

Date/Publication 2026-11-08 17:50:25 UTC

Contents

avc . . . 3
cabgkul . . . 4
coef.hazard . . . 5
hazard . . . 6
hzr_argument_mapping . . . 18
hzr_bootstrap . . . 19
hzr_calibrate . . . 24
hzr_clamp_prob . . . 25
hzr_competing_risks . . . 26
hzr_deciles . . . 28
hzr_decompos . . . 30
hzr_decompos_g3 . . . 32
hzr_evaluate . . . 34
hzr_gof . . . 35
hzr_kaplan . . . 38
hzr_log1mexp . . . 40
hzr_log1pexp . . . 41
hzr_nelson . . . 42
hzr_phase . . . 43
hzr_phase_cumhaz . . . 47
hzr_phase_hazard . . . 48
hzr_read_outhaz . . . 50
hzr_repeated_events . . . 51
hzr_stepwise . . . 52
hzr_theta_names . . . 60
hzr_translate_sas . . . 61
is_hzr_phase . . . 64
omc . . . 65
predict.hazard . . . 65
predict.hzr_outhaz . . . 71
print.hzr_calibrate . . . 74
print.hzr_deciles . . . 74
print.hzr_gof . . . 75

print.hzr_kaplan 76
print.hzr_sas_job 76
stepwise_trace 77
summary.hazard 78
tga 79
uslife2023 80
valves 81
vcov.hazard 83

Index **84**

---

avc *AVC: Atrioventricular Canal Repair*

Description

Survival data for 310 patients who underwent repair of atrioventricular septal defects (congenital heart disease) at the Cleveland Clinic between 1977 and 1993. Exhibits two identifiable hazard phases: an early post-operative risk and a constant late phase.

Usage

```
avc
```

Format

A data frame with 310 rows and 11 variables:

study Patient identifier

status NYHA functional class (1–4). A covariate, not a censoring status: the event indicator is `dead`.

inc_surg Surgical grade of AV valve incompetence

opmos Date of operation (months since January 1967)

age Age at repair (months)

mal Malalignment indicator (0/1)

com_iv Interventricular communication indicator (0/1)

orifice Associated cardiac anomaly indicator (0/1)

dead Death indicator (1 = dead, 0 = censored)

int_dead Follow-up interval to death or last contact (months)

op_age Interaction term: opmos x age

Source

Blackstone, Naftel, and Turner (1986) doi:10.1080/01621459.1986.10478314. Cleveland Clinic Foundation.

See Also

vignette("fitting-hazard-models"), vignette("prediction-visualization")

Other datasets: cabgkul, omc, tga, uslife2023, valves

Examples

```
data(avc)
avc <- na.omit(avc)

# Kaplan-Meier survival
km <- survival::survfit(survival::Surv(int_dead, dead) ~ 1, data = avc)
plot(km, xlab = "Months after AVC repair", ylab = "Survival",
     main = "AVC: Kaplan-Meier survival estimate")


# Two-phase hazard fit (early CDF + constant -- what AVC supports)
fit <- hazard(
  survival::Surv(int_dead, dead) ~ 1, data = avc,
  dist = "multiphase",
  phases = list(
    early    = hzr_phase("cdf", t_half = 0.5, nu = 1, m = 1),
    constant = hzr_phase("constant")
  ),
  fit = TRUE, control = list(n_starts = 5, maxit = 1000)
)
summary(fit)
```

---

cabgkul *CABGKUL: Primary Isolated Coronary Artery Bypass Grafting (KU Leuven)*

Description

Survival data for 5,880 patients who underwent primary isolated CABG at KU Leuven, Belgium, between 1971 and July 1987. The simplest dataset structure (intercept-only, right-censored) with large sample size exercising all three temporal hazard phases.

Usage

```
cabgkul
```

Format

A data frame with 5880 rows and 2 variables:

int_dead Follow-up interval to death or last contact (months)

dead Death indicator (1 = dead, 0 = censored)

Source

KU Leuven cardiac surgery registry. Primary benchmark dataset for C binary parity testing.

See Also

`vignette("fitting-hazard-models")`

Other datasets: `avc`, `omc`, `tga`, `uslife2023`, `valves`

Examples

```
data(cabgkul)

# Kaplan-Meier survival
km <- survival::survfit(survival::Surv(int_dead, dead) ~ 1, data = cabgkul)
plot(km, xlab = "Months after CABG", ylab = "Survival",
     main = "CABGKUL: Kaplan-Meier survival (n = 5,880)")


# Single-phase Weibull fit with parametric overlay
fit <- hazard(survival::Surv(int_dead, dead) ~ 1, data = cabgkul,
              dist = "weibull", theta = c(mu = 0.10, nu = 1.0), fit = TRUE)
t_grid <- seq(0.01, max(cabgkul$int_dead) * 0.9, length.out = 200)
surv   <- predict(fit, newdata = data.frame(time = t_grid),
                  type = "survival")
plot(km, xlab = "Months after CABG", ylab = "Survival",
     main = "CABGKUL: Weibull vs. Kaplan-Meier")
lines(t_grid, surv, col = "blue", lwd = 2)
legend("bottomleft", c("KM", "Weibull"), col = c("black", "blue"),
       lty = 1, lwd = c(1, 2))
```

---

coef.hazard *Extract coefficients from hazard model*

Description

Extract coefficients from hazard model

Usage

```
## S3 method for class 'hazard'
coef(object, ...)
```

Arguments

`object`	A `hazard` object.
`...`	Unused; for S3 compatibility.

Value

A named numeric vector of fitted parameter estimates, or NULL if the model has not been fitted (fit = FALSE).

Examples

```
set.seed(1)
fit <- hazard(time = rexp(30, 0.5), status = rep(1L, 30),
              theta = c(0.3, 1.0), dist = "weibull", fit = TRUE)
coef(fit)
```

---

hazard *Build and optionally fit a hazard model*

Description

Creates a hazard object and optionally fits it via maximum likelihood. This mirrors the argument-oriented workflow of the legacy HAZARD C/SAS implementation: supply starting values in theta and the function will optimize to produce fitted estimates.

Usage

```
hazard(
  formula = NULL,
  data = NULL,
  time = NULL,
  status = NULL,
  time_lower = NULL,
  time_upper = NULL,
  x = NULL,
  time_windows = NULL,
  theta = NULL,
  dist = "weibull",
  phases = NULL,
  fit = FALSE,
  weights = NULL,
  control = list(),
  objective = c("likelihood", "sas"),
  ...
)
```

Arguments

formula	Optional formula with a Surv() object on the left. Right-censored (Surv(time, status)), left-censored (Surv(time, event, type = "left")), interval-censored (type = "interval" or "interval2") and counting-process (Surv(start, stop,

event)) forms are all accepted. Surv() codes censoring status with different integers than this package does; the formula path translates them, so write Surv()'s codes here. A plain status vector takes this package's codes; a Surv passed as status is translated the same way as here. Every distribution carries its own intercept: its baseline parameter, the one the covariates add to, already plays that role, so the design never has one: removing it (~ 0 + age, ~ age - 1) is ignored with a warning, and builds the design of ~ age. A . on the right-hand side stands for every column of data that the Surv() term does not use, as in survival::coxph(). When provided, overrides direct time/status/x arguments and extracts from data. Example: hazard(Surv(time, status) ~ x1 + x2, data = df, dist = "weibull", fit = TRUE).

data — Optional data frame. On the formula path it supplies the model frame, and weights is evaluated in its scope. On the vector path time, status, time_lower, time_upper and weights are evaluated in its scope, the way base::subset() and base::transform() do: a bare column name resolves to that column, and anything that is not a column (df$col, a local vector, a literal) falls through to the calling environment. A column of the same name as a caller variable wins, and because that silently discards the caller's vector (the way a wrapper forwarding its own argument by name does), such a name raises a warning naming the symbol and the argument. The warning reads the names written in the expression. When the argument is the name alone, it says the column was used. When the name is part of a larger expression, it does not say which value was read: the expression may never evaluate the name (an unused function argument), may rebind it first (a loop variable, an assignment) or may evaluate it somewhere else (with()), and the warning cannot tell. A name chosen at run time, as in get(nm) or eval(as.name(nm)), resolves the same way, column first, as it does in stats::lm(), but is not checked; the vector path has behaved so since 1.2.2. No second evaluation is made to compare the two values, because evaluating an expression such as runif(n) twice gives two different vectors. Masked arguments are validated like any other, so an NA in a masked column errors: an NA count on the SAS ICENSOR path reaches weights and stops with 'weights' must be non-negative and finite. A named element of data that is a **function** is refused, on both paths, as stats::lm() refuses it. Because the mask sits in front of the calling frame, such an element would be called in place of the function an expression names – weights = rep(1, n) calling a rep held in data – and the fit would change with nothing to show for it. An S4 generic and a reference-class generator are functions for this purpose. A list-column of functions is a list, not a function, and is unaffected, as is an element with no name, which no expression can look up – but an NA_character_ name is refused, because R binds such an element under the symbol `NA` and a call reaches it. Remove the element: for a vector argument or weights, define the helper in the calling environment; for one used inside the Surv() response, compute the value into a data column first, since the response is evaluated without the formula's environment.

time — Numeric follow-up time vector. A row whose time is 0 is dropped before fitting, whatever its status, as PROC HAZARD drops it (TIME <= 0 is inadmissible there). The time tested is the row's upper bound: time for an exact or right-censored row, time_upper for a left- or interval-censored one; a lower

bound or entry time of 0 is admissible. `hazard()` warns with the count (class `"hzr_time_zero_dropped"`), and every row stored on the fit is what remains. `fit$data$dropped_time_zero` is the count and `fit$data$dropped_time_zero_rows` their positions among the rows given. On the formula interface the response and design are then built on the retained rows, as if the dropped ones had not been given, so a data-dependent term such as `scale(age)` uses the retained rows. Two inputs are refused because they cannot follow the rows: a response whose values change once the rows are dropped, such as `Surv(time - min(time), status)`, and a formula that reads a per-row value from outside `data`. The second is read two ways. A variable named in the formula is refused when it is not a column of `data`. A helper that reads an outside vector inside its body, such as `function(a) a + g[seq_along(a)]`, names no such variable, and on the retained rows would pair them with the first elements of `g`. So the formula's variables are evaluated on every row in order, and again with the retained rows first, in their order, as the fit sees them; a variable that then gives a retained row a different value is refused (#590). Both times every row is present, so `scale()`, `poly()`, `ns()` and factors pass, including a factor whose levels are taken in order of appearance. A term that cannot be evaluated on the dropped rows, such as `poly()` over an NA only they carry, is evaluated with those rows replaced by a copy of a retained one. A term that depends on the order of the rows, such as `cumsum(age)`, or is random, such as `jitter(age)`, is refused unless the dropped rows are the last ones. A helper that depends on how many rows it is given, such as `tail(g, length(a))` or `a * mean(g[seq_along(a)])`, is not caught, because both evaluations see every row. Put such a vector in `data`, where it is dropped with its rows. A caller-supplied row-aligned input whose row count differs from `time`'s is refused before any row is dropped: `x` and `weights` on the `time =/status =` interface, and `data` wherever a formula reads it row by row. (On the formula interface the design is not the caller's; it is rebuilt from the retained rows of `data`.)

`status` Numeric or logical event indicator vector, or a `survival::Surv()` object. A `Surv` is read by its `type`, exactly as the formula interface reads it, and a `time`, `time_lower` or `time_upper` that disagrees with it is an error.

`time_lower` Optional numeric vector whose role depends on `status`. Supplying it explicitly is **not** a no-op.

- `status == 2` (interval-censored): the lower bound of the censoring interval, defaulting to `time`. Every `dist` reads it this way.
- `status %in% c(0, 1)` (right-censored or event): the counting-process **entry time** when `0 < time_lower < time`, so the row contributes `H(time) - H(time_lower)`. Every `dist` reads it this way. A value of `0`, or equal to `time`, means no entry time, and left `NULL` the entry time is `0`.
- `status == -1` (left-censored): not used; the bound is `time_upper`.

`time_lower = time` on the status 0 and 1 rows is the mixed-interval layout: exact and right-censored rows carry their own time, and only the status-2 rows carry a real lower bound. A subject cannot enter the risk set after the moment it leaves, so `hazard()` stops with an error when a status 0 or 1 row has `time_lower > time`, as SAS HAZARD rejects a start time after the exit time. It also stops when rows with `time_lower == time > 0` sit beside rows with gen-

uine entry times: in counting-process data those are zero-length epochs, which `hzr_repeated_events()` can emit and which must be adjusted first.

`time_upper` Optional numeric upper bound vector for censoring intervals. Used when `status %in% c(-1, 2)`; defaults to `time` if NULL.

`x` Optional design matrix (or data frame coercible to matrix). Unlike `time` and the other vector arguments, `x` is **not** data-masked: `hazard(data = df, time = tt, x = age)` errors with `object 'age' not found` where `time = tt` resolves, so write `x = df[["age"]]` or use the formula interface.

`time_windows` Optional numeric vector of strictly positive cut points for piecewise time-varying coefficients. When provided, each predictor column in `x` is expanded into one column per time window so each window gets its own coefficient.

`theta` Numeric starting values for optimization: the shape parameters, then one coefficient per column of the design matrix. Required when `fit = TRUE` with a single-distribution `dist`; optional for `dist = "multiphase"`, which assembles its own starting values from `phases` when `theta` is `NULL`.

`dist` Character baseline distribution label (default `"weibull"`). One of `"weibull"`, `"exponential"`, `"loglogistic"`, `"lognormal"`, or `"multiphase"`. The single-distribution families differ in the *shape* the hazard traces over time; `"multiphase"` builds an additive N-phase hazard and requires `phases`. See the **Baseline distributions** section for what each means and when to use it.

`phases` Optional named list of `hzr_phase()` objects specifying the phases for a multiphase model (`dist = "multiphase"`). Names must be unique, and `"total"` and `"time"` are reserved: they label the summed column and the column of prediction times in the decomposed output of `predict.hazard()`, so a phase of either name is rejected. See Examples.

`fit` Logical; if TRUE, fit the model via maximum likelihood (default FALSE).

`weights` Optional numeric vector of observation weights (non-negative). Each observation's log-likelihood contribution is multiplied by its weight. Use for severity-weighted repeated events. Default `NULL` (unit weights). Implements the SAS `WEIGHT` statement. With `data`, a name is looked up among its columns first and then in the calling environment, on both interfaces. `stats::lm()` also looks in `data` first, but then in the formula's environment rather than the caller's, so a formula built inside another function does not bring that function's variables with it here. See `data` for the warning raised when a name is both. `weights` is evaluated over every row given, so an expression that depends on the rows – `weights = w / mean(w)` – includes any row later dropped for time 0 (see `time`); compute it in `data` on the rows you intend to fit if that matters.

`control` Named list of control options (see Details).

`objective` Which interval-censored contribution the multiphase likelihood accumulates. `"likelihood"` (default) uses the interval probability $\log(S(l) - S(u))$. `"sas"` reproduces what `PROC HAZARD` accumulates: the event-density term with the instantaneous hazard replaced by the interval-mean hazard over $(l, u]$. Applies only to `dist = "multiphase"`; exact-event and right-censored rows are unaffected either way.

`"sas"` requires data it can represent: no left-censored rows (`PROC HAZARD` has no left-censoring statement) and a positive width on every interval-censored row

(the interval-mean hazard divides by $u-l$). Both are properties of the data rather than of the fit, so they are checked when the argument is supplied, including under `fit = FALSE`, which therefore stops rather than returning an unusable object.

... Additional named arguments retained for parity with legacy calling conventions.

Details

Control parameters:

- `maxit`: Maximum iterations of the quasi-Newton (BFGS) optimizer (default 1000), applied to each start. A fit whose optimizer reports convergence but fails SAS's gradient test is continued with `stats::nlm()` under the same limit (see "Convergence"), so raising `maxit` lets that continuation run further too. The Nelder-Mead warm-up that a multiphase fit with fixed parameters may run first has its own limit, which `maxit` does not change. It must be a single finite number of at least 1 (a fraction is truncated, as PROC HAZARD truncates `MAXITER`): anything else stops `hazard()` at once, fitted or not (#541). A fit with no iterations is its starting values, not an estimate; to evaluate a model at parameters you supply, use `hzr_evaluate()`.
- `n_starts`: Number of optimization starts for multiphase fits (default 5). Each start after the first offsets the initial values. The offsets are computed by the package without using R's random number stream, so a multiphase fit is reproducible without `set.seed()` and neither reads nor changes that stream.
- `start_seed`: Seed selecting the ensemble of multi-start offsets (default 3). Any whole number within integer range, negative included. Fits are reproducible at any value; vary it to probe a different set of starts when a fit is suspected of sitting in a local optimum, and compare the resulting `objective` values. A fractional value is rejected rather than truncated, since `3.9` and `3` would otherwise select the same ensemble without saying so.
- `phase_share_tol`: Threshold for the multiphase identifiability warning (default 1e-8). A phase is reported as having left the model when it supplies less than this share of the cumulative hazard at every observed time (it never started; neither its `mu` nor its shape is identified), or when its contribution varies by less than this relative amount across them (it finished before the first observation and acts as a constant offset; `mu` stays identified, the shape parameters do not). A third condition is a property of the observed times rather than of any phase: when their own relative range falls below this threshold and no phase's contribution varies across them, they separate no phase from any other and none of the shapes is identified. That verdict is withheld when the fit supplies counting-process entry times or interval bounds that add enough evaluation points the measure did not see. "Enough" is counted, not assumed: with the event times tied, each added point supplies one more functional of the parameters, and one more than the number of added points must reach the number of free parameters. A zero entry time, or a bound equal to an event time, adds nothing and does not count. The count is deliberately conservative: it does not credit the extra functional that a zero entry time makes separately observable, so a fit can be warned about while being identified. For the same reason the per-phase measures can overstate what an interval-censored or left-truncated fit loses, since a phase flat across the event times may still be identified through the bounds; the saturated warning says so where such points exist, rather than claiming the likelihood is unchanged (#228). The measured shares are kept on the fit as `fit$fit$phase_share`. Raise it to catch marginal phases, set it to 0 to silence the check.
- `reltol`: Relative convergence tolerance on the objective the optimizer minimizes, the negative log-likelihood (default 1e-5). The stored `fit$fit$objective` has the opposite sign (see

Value). BFGS stops when an iteration reduces it by less than `reltol * (|objective| + reltol)`, so the stopping gap grows with the size of the log-likelihood: about 0.0024 at a log-likelihood of -240. On a flat surface plain BFGS can stop that far short of the optimum and still report convergence, so `hazard()` then applies SAS/C HAZARD's relative-gradient test and, when the stop fails it, continues with `stats::nlm()`; see the "Convergence" section.

- `conserve`: Apply Conservation of Events (`dist = "multiphase"` **only**; default `TRUE`). CoE counts exact events, so it is **automatically disabled** whenever any `status` falls outside {0, 1} (which interval or left censoring guarantees) and whenever the model has fewer than two phases. On a fitted multiphase object the outcome is recorded next to the request, both fields living under `fit$spec$control`:
  - `fit$spec$control$conserve_applied`: logical, whether CoE was actually applied;
  - `fit$spec$control$conserve_disabled_reason`: one of `"not_requested"`, `"unsupported_censoring"`, `"single_phase"`, `"no_events"`, `"setup_failed"`, or `NA` when CoE was applied.

  Read `fit$spec$control$conserve_applied`, not `fit$spec$control$conserve`: the latter says only what you asked for.
- `shape_param_count`: The number of baseline parameters at the front of `theta`: the scale and any shape parameters, so 2 for `"weibull"` and 1 for `"exponential"`. For a single-distribution model only. The fit does not use it: `theta` is checked against the distribution's own count, and so are `hzr_stepwise()`, its refits and its score test. Only an internal helper that writes a SAS input deck for parity checks reads it. A multiphase fit derives its own layout, so nothing reads it there.

The elements above are accepted without a warning: `maxit` and `reltol` for every model, `shape_param_count` for a single-distribution model, and `n_starts`, `start_seed`, `phase_share_tol` and `conserve` for `dist = "multiphase"` (#376). Any other element draws one warning that names it and says why it has no effect, and the fit proceeds unchanged, as `stats::optim()` does for unknown `control` names. The element is dropped before the fit, so a name such as `n_starts_extra` cannot be read as `n_starts`, and `fit$spec$control` keeps none of the ignored elements. That covers:

- a name no fit reads, such as the misspelling `n_startz`, and an unnamed element;
- `abstol` (read only by a bounded optimizer no fit uses), `method`, `condition`, `nocov` and `nocor`, which earlier versions documented as accepted without reading them;
- `fix` and `quasi`, which no fit has ever read. A fit given `fix` was never constrained, so results obtained with it may be affected; hold a parameter with `hzr_phase(fixed = )` on a multiphase phase. A single-distribution model has no mechanism for fixing a parameter.
- a multiphase element such as `n_starts` given to a single-distribution fit, and `shape_param_count` given to a multiphase one.

No name in `control` is an error. A bad `maxit` stops `hazard()` at once (see above); a bad value for another element the fit reads still stops a fit (`fit = TRUE`) where it is read. `hzr_stepwise()` and `hzr_bootstrap()` pass `control` to every candidate refit, and an error there would count as a failed candidate, so a screen would report success having tested nothing.

SAS `PROC HAZARD` options with no `control` equivalent: `NOCOV` and `NOCOR` only suppress printed output, and `hazard()` prints nothing until asked. `CONDITION=` stops SAS's optimizer on a condition-number test that `hazard()` does not have. `QUASI` is `hazard()`'s optimizer already: the fits it runs use BFGS (a multiphase fit may run a Nelder-Mead warm-up first, and a stop that fails SAS's gradient test continues with `stats::nlm()`). SAS jobs often write `STEEPEST QUASI` together, steepest

descent first; **there is no steepest-descent option and no two-stage strategy**. The multiphase likelihood is multimodal, so a different descent path can land on a different optimum: a fit translated from a job using `STEEPEST` may not reproduce SAS's estimates, and `hzr_translate_sas()` records the keyword as untranslated rather than dropping it.

Censoring status coding:

- 1: Exact event at time
- 0: Right-censored at time
- -1: Left-censored with upper bound at time_upper \(or time\)
- 2: Interval-censored in the interval \(time_lower, time_upper\)

Time-varying coefficients:

- If `time_windows` is supplied, predictors are expanded to piecewise window interactions so each window has its own coefficient vector.
- This is implemented as design-matrix expansion, so the existing likelihood engines remain unchanged.

Value

An object of class `hazard`, a named list with components: `call` (the matched call), `spec` (model specification: `dist`, `control`, `time_windows`, `phases`), `data` (input data: `time`, `status`, `x`, `weights`, etc.), `fit` (optimisation results: `theta`, `objective`, the maximized log-likelihood, so larger is better, or PROC HAZARD's objective under `objective = "sas"` (`summary()` reports which, as its `objective`); `converged`, `se`, `vcov`, `counts`, `message`, and `rel_gradient`, `rel_gradient_reason` and `polish_code`, the SAS/C acceptance test described under "Convergence"; all `NULL` when `fit = FALSE`; multiphase fits add `starts`, one row per optimisation start with its `status` (`"ok"`, `"nonconverged"`, `"infeasible"`, `"nonfinite"` or `"error"`), `objective` (on the same scale as the fit's, so the best start has the largest; `NA` unless the start reached a point where the likelihood is defined, and recorded as the optimizer returned it, before any Conservation of Events adjustment to the conserved phase's scale), `convergence` (the `optim` code, `0` for success), whether it was the `best` and so the reported fit, and the `message` of any error. A start that stops at `maxit` has a finite `objective` and can win the selection, so `status` distinguishes it from one that converged. Every fit carries `weak`, which takes one of three values: a list, when the likelihood is near-flat along a parameter combination, giving the `params` spanning that direction, their squared loadings (`weights`), the strongest pairwise `correlation` among them, the Hessian `rcond` and `n_directions`, the number of near-flat directions found (when a single parameter carries the direction on its own, the list has one `params` entry, `single = TRUE`, its `estimate`, and `correlation = NA`, and `se_metric`, its standard error on the log scale; only a g3 shape, `gamma`, `alpha` or `eta`, or a multiphase phase's `log_mu`, `log_t_half` or `log_tau`, is named this way, the last three when their standard error exceeds 10 log units, and with `weights = NA`); `NULL` when the fit was examined and is well identified; and `NA` when the check could not run because no usable Hessian was available, which includes an unfitted object, an install without the suggested **numDeriv**, a Hessian that is not positive-definite (unless a ridge was found, which is reported), and an ill-conditioned fit in which an estimated parameter has no finite variance and no ridge was found among the others, since that parameter was not examined. Test with `is.list(fit$fit$weak)`, not `!is.null()`: the `NA` case has not been examined and must not be read as a clean result), `engine` (implementation tag, `"native-r-m2"`), and two fields recording what the fit did not do: `degraded`, a character vector of the steps not

performed, in the fixed order "fitting", "standard_errors", "conserved_phase_variance", "weak_direction_check", "boundary_check", "conservation_of_events", and empty when nothing was lost; and degraded_causes, a character vector with the same names giving the reason for each. print() and summary() always show them as a "Not done in this run" block, which reads "none" when nothing was lost. fitfitweak is NA exactly when "weak_direction_check" is listed.

fitfitboundary is its sibling and takes the same three states, for phases whose fit sits where their parameterisation breaks down: NULL when the check ran and found nothing, a list of records when it found something, and NA when it did not run — and it is NA exactly when "boundary_check" is listed in degraded, with the reason in degraded_causes. Each record carries mechanism, phase, parameter and a printable detail. The mechanisms are "unbounded_phase", a "hazard" phase whose t_half is below the first observed time; "phase_discontinuity", a "cdf" or "hazard" phase whose shape has collapsed to a step the observed times cannot resolve (it can lie inside the data); and two made at setup, before the fit: "g3_alpha_one" (a g3 phase with alpha fixed at 1, re-expressed as PROC HAZARD does, see hzr_phase()) and "g3_fixge2_alpha_start" (a free alpha start moved to 2/3 under constraint = "eta_gamma"); and two made after it. "g3_corner_supremum" is a g3 phase under constraint = "eta_gamma" whose estimated gamma converged below 1000 although the log-likelihood is higher at a much larger gamma, with any fixed tau or alpha held. The estimate is then not the maximum-likelihood one, and the supremum may lie at gamma = Inf. A fixed gamma, and a fit with left- or interval-censored rows, are not examined. "coe_no_events_left" is a fit under Conservation of Events whose log-likelihood is higher with the conserved phase's scale sent to zero, where no events remain for that phase. Both records also carry gain and the higher point as certificate_theta with its certificate_loglik (the corner record adds gamma_hat); the estimates are not changed. "coe_phase_vanished" is a fit under Conservation of Events that has itself run to that boundary, or beside it: the conserved phase's largest share of the cumulative hazard is below control$phase_share_tol. Its record carries share and tol, and warned_by = "phase_share": the identifiability warning has already reported the phase, so this record raises no warning of its own. A fit can carry several records, in no guaranteed order, so select them by mechanism. Only rows the likelihood reads count as observed times. A fit with any record that has no warned_by raises one warning whose classes are "hzr_" plus each such record's mechanism, all inheriting "hzr_boundary", so one handler catches the whole family.

The fitted model

For dist = "multiphase" the cumulative hazard, instantaneous hazard, and survival are

$$H(t \mid \mathbf{x}) = \sum_{j=1}^{J} \mu_j(\mathbf{x})\, \Phi_j(t), \qquad h(t \mid \mathbf{x}) = \sum_{j=1}^{J} \mu_j(\mathbf{x})\, \varphi_j(t), \qquad S(t \mid \mathbf{x}) = \exp\bigl(-H(t \mid \mathbf{x})\bigr)$$

where $\mu_j(\mathbf{x}) = \exp(\alpha_j + \mathbf{x}_j^\top \beta_j)$ and the temporal shapes Φ_j, φ_j are set by each phase's type (see hzr_phase()). The proportional-hazards single-phase families ("weibull", "exponential") are the special case $J = 1$, with covariates acting multiplicatively on one temporal shape. The "loglogistic" (proportional-odds) and "lognormal" (accelerated-failure-time) families place covariates differently (on the odds of failure and the log-time location, respectively), so they are separate parameterizations, not special cases of this additive form. Parameters are estimated on an unconstrained internal scale (e.g. $\log \mu$, $\log t_{1/2}$) and transformed back for reporting; see vignette("mf-mathematical-found

Convergence

The optimizer stops when an iteration improves the log-likelihood by less than `control$reltol` relative to its size, and on a flat ridge that can happen well short of the maximum. SAS/C HAZARD accepts an optimum only on a different test, the relative gradient $\max_i |g_i| \max(|x_i|, 1) / \max(|\ell|, 1) \leq \epsilon^{1/3}$, about 6e-6, and `hazard()` applies it too: when the optimizer reports convergence and the test fails, the fit is continued with `stats::nlm()` at SAS's tolerances, and the continued point is kept if it improves the log-likelihood.

Every fit records the result in `fit$fit$rel_gradient` and, when the continuation improved the fit, its termination code in `fit$fit$polish_code`. `print()` and `summary()` show both. `rel_gradient` is `NA` when the test was not applied (the optimizer did not report convergence) or the gradient cannot be evaluated at the estimates; `NA` is never reported as a pass. Neither is it always a failure: some routes to it, such as a non-converged stop, do say the estimates are unreliable, while others, such as a finite-difference score that needed a point where the log-likelihood is not usable, say nothing against them. Which route it took is recorded in `fit$fit$rel_gradient_reason`, `NA_character_` when the test did run, and `print()` and `summary()` show it. Under Conservation of Events the conserved scale is re-solved from the other parameters at every step, as SAS/C does, and the score used by the optimizer, the continuation and the test includes how that scale moves with them. A fit can still end with the test not met, under Conservation of Events or without it. A warning is raised only for code 4, the iteration limit (raise `control$maxit`), and code 5, where the log-likelihood kept rising along some direction and the model may have no maximum. Codes 2 and 3, where SAS/C prints a caution, are recorded without one, except for the single-distribution fits: there a converged fit with a relative gradient above 1e-3 that did not stop on code 4 or 5 (in practice code 2 or 3, or no code, which is recorded only when the continuation's point was kept: none is when it found no better point, `stats::nlm()` raised an error, or its minimum was not finite) warns, with class `"hzr_possible_false_maximum"`, that the fit may not be a maximum, and suggests other starting values or centring or rescaling the covariates. Badly scaled covariates can put a good fit above 1e-3, where a restart does not help and rescaling does. `hzr_bootstrap()` counts the replicates in which any fit (the base refit, a stepwise refit or the final fit) meets the same rule, and warns once. Such a stop can be a false maximum far below the best one from an ordinary start, and the gradient test alone cannot always tell it from a good fit, so `converged` is left as it is. The warning is not a guarantee: a fit can stop short of its maximum with a relative gradient below 1e-3 and raise nothing. The test is relative to the size of the log-likelihood, so a fit that meets it is within SAS's tolerance of the maximum, not exactly at it.

The optimizer treats a score it cannot use as zero, so it can stop on those zeros far from the maximum. For the single-distribution fits (exponential, Weibull, lognormal and loglogistic), a stop where the score at the estimates is not finite, or could not be computed, is therefore reported as `converged = FALSE`, with a warning of class `"hzr_unverified_convergence"`; `rel_gradient_reason` names which.

Baseline distributions

The `dist` argument selects the parametric form of the baseline hazard. The four single-distribution families differ in the *shape* the hazard traces over follow-up; choose by what the risk is expected to do over time. `"multiphase"` is the general additive model that lets several such shapes coexist.

`"weibull"`**: monotone rising or falling hazard (default)** The workhorse parametric model: $H(t \mid \mathbf{x}) = (\mu t)^{\nu} \exp(\eta)$, with hazard $h \propto t^{\nu-1}$. The single shape ν makes risk increase over time ($\nu > 1$), decrease ($\nu < 1$), or stay flat ($\nu = 1$). Use it as the default when a single

monotone trend describes the hazard. The scale μ is the baseline at $\mathbf{x} = 0$, so a covariate far from zero (a calendar year, an age in days) can push it beyond what a number can hold: it is then reported as `Inf`, as 0, or as a value too small to keep its digits, with a warning of class `"hzr_unrepresentable_scale"`. The fit keeps $\log \mu$ and its standard error, which `summary()` shows as a `log(mu)` row and `predict()` reads; `coef()` shows μ as it is, and `vcov()` gives `NA` for it. Centering or rescaling the covariate reports μ itself, without changing the model.

`"exponential"`**: constant hazard** The memoryless special case $\nu = 1$: a time-invariant baseline rate, $H(t \mid \mathbf{x}) = \mu t \exp(\eta)$. Use it when the event rate does not change with follow-up time (the constant background risk also appears as the `"constant"` phase in a multiphase model).

`"loglogistic"`**: unimodal (rise-then-fall) hazard** A log-logistic proportional-odds form (covariates act multiplicatively on the odds of failure, $\exp(\eta)$, not as an AFT time shift) whose hazard rises to a single peak and then declines when the shape exceeds 1 (and is monotone decreasing otherwise), with heavier tails than the log-normal. Use it when risk climbs to an early peak and then eases off.

`"lognormal"`**: early-peaking, resolving hazard** An accelerated-failure-time form in which $\log$ time is Gaussian; the hazard rises to an early peak and then decays toward zero. Use it for risk that is concentrated early and resolves over time.

`"multiphase"`**: additive N-phase hazard** Sums several phase shapes into one model, $H = \sum_j \mu_j(\mathbf{x})\Phi_j(t)$, so the overall hazard can fall, level off, and rise again within one fit. Requires `phases`; see `hzr_phase()` for the available phase shapes. This is the form that reproduces the classic C/SAS HAZARD models.

Note

`objective = "sas"` exists to reproduce legacy `PROC HAZARD` runs and **must not be used for new analyses**. It is a density, not a probability: it is inconsistent for wide intervals, where the two forms differ materially (22 log-likelihood units on the esophagectomy reference fit). The default is the statistically correct interval likelihood. See `inst/dev/SAS-INTERVAL-OBJECTIVE-DESIGN.md` for the derivation and the four-reference evidence.

References

Blackstone EH, Naftel DC, Turner ME Jr. The decomposition of time-varying hazard into phases, each incorporating a separate stream of concomitant information. *J Am Stat Assoc.* 1986;81(395):615–624. doi:10.1080/01621459.1986.10478314

Rajeswaran J, Blackstone EH, Ehrlinger J, Li L, Ishwaran H, Parides MK. Probability of atrial fibrillation after ablation: Using a parametric nonlinear temporal decomposition mixed effects model. *Stat Methods Med Res.* 2018;27(1):126–141. doi:10.1177/0962280215623583

See Also

`predict.hazard()` for survival/cumulative-hazard predictions, `summary.hazard()` for model summaries, `hzr_phase()` for specifying multiphase temporal shapes.

Vignettes with worked examples: `vignette("fitting-hazard-models")`: single-phase through multiphase fitting, `vignette("prediction-visualization")`: prediction types and decomposed hazard plots, `vignette("inference-diagnostics")`: bootstrap CIs and model diagnostics.

Examples

```
# -- Univariable Weibull ------------------------------------------------
set.seed(1)
time   <- rexp(50, rate = 0.3)
status <- sample(0:1, 50, replace = TRUE, prob = c(0.3, 0.7))
fit <- hazard(time = time, status = status,
              theta = c(0.3, 1.0), dist = "weibull", fit = TRUE)
summary(fit)

# -- Formula interface with covariates ----------------------------------
set.seed(1001)
n   <- 180
dat <- data.frame(
  time   = rexp(n, rate = 0.35) + 0.05,
  status = rbinom(n, size = 1, prob = 0.6),
  age    = rnorm(n, mean = 62, sd = 11),
  nyha   = sample(1:4, n, replace = TRUE),
  shock  = rbinom(n, size = 1, prob = 0.18)
)

fit2 <- hazard(
  survival::Surv(time, status) ~ age + nyha + shock,
  data    = dat,
  theta   = c(mu = 0.25, nu = 1.10, beta1 = 0, beta2 = 0, beta3 = 0),
  dist    = "weibull",
  fit     = TRUE,
  control = list(maxit = 300)
)
summary(fit2)


# -- Parametric survival with Kaplan-Meier overlay ----------------
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)

  # Parametric curve on a fine grid at median covariate profile
  t_grid   <- seq(0.05, max(dat$time), length.out = 80)
  curve_df <- data.frame(
    time = t_grid, age = median(dat$age), nyha = 2, shock = 0
  )
  curve_df$survival <- predict(fit2, newdata = curve_df,
                               type = "survival") * 100

  # Kaplan-Meier empirical overlay
  km    <- survival::survfit(survival::Surv(time, status) ~ 1, data = dat)
  km_df <- data.frame(time = km$time, survival = km$surv * 100)

  ggplot() +
    geom_step(data = km_df, aes(time, survival, colour = "Kaplan-Meier")) +
    geom_line(data = curve_df, aes(time, survival,
                                   colour = "Parametric (Weibull)")) +
    scale_colour_manual(
```

```
      values = c("Parametric (Weibull)" = "#0072B2",
                 "Kaplan-Meier"         = "#D55E00")
    ) +
    scale_y_continuous(limits = c(0, 100)) +
    labs(x = "Months after surgery", y = "Freedom from death (%)",
         colour = NULL) +
    theme_minimal() +
    theme(legend.position = "bottom")
}



# -- Multiphase model (two phases) ---------------------------------
fit_mp <- hazard(
  survival::Surv(time, status) ~ 1,
  data   = dat,
  dist   = "multiphase",
  phases = list(
    early = hzr_phase("cdf", t_half = 0.5, nu = 2, m = 0,
                      fixed = "shapes"),
    late  = hzr_phase("cdf", t_half = 5,   nu = 1, m = 0,
                      fixed = "shapes")
  ),
  fit     = TRUE,
  control = list(n_starts = 5, maxit = 1000)
)
summary(fit_mp)

# -- Per-phase decomposed cumulative hazard ------------------------
if (requireNamespace("ggplot2", quietly = TRUE)) {
  t_grid <- seq(0.01, max(dat$time), length.out = 100)
  decomp <- predict(fit_mp, newdata = data.frame(time = t_grid),
                    type = "cumulative_hazard", decompose = TRUE)

  df_long <- data.frame(
    time = rep(decomp$time, 3),
    cumhaz = c(decomp$total, decomp$early, decomp$late),
    component = rep(c("Total", "Early (cdf)", "Late (cdf)"),
                    each = nrow(decomp))
  )
  df_long$component <- factor(df_long$component,
    levels = c("Total", "Early (cdf)", "Late (cdf)"))

  ggplot2::ggplot(df_long,
    ggplot2::aes(x = time, y = cumhaz, colour = component,
                 linewidth = component)) +
    ggplot2::geom_line() +
    ggplot2::scale_colour_manual(values = c(
      "Total" = "black", "Early (cdf)" = "#0072B2",
      "Late (cdf)" = "#D55E00"
    )) +
    ggplot2::scale_linewidth_manual(values = c(
      "Total" = 1.2, "Early (cdf)" = 0.6, "Late (cdf)" = 0.6
```

```
    )) +
    ggplot2::labs(
      x = "Time", y = "Cumulative hazard H(t)",
      colour = NULL, linewidth = NULL,
      title = "Multiphase decomposition: early + late"
    ) +
    ggplot2::theme_minimal() +
    ggplot2::theme(legend.position = "bottom")
}
```

`hzr_argument_mapping` *Legacy HAZARD to TemporalHazard argument mapping*

Description

Returns a formal mapping table that defines how legacy SAS HAZARD/C-style inputs map to `hazard(...)` arguments in this package.

Usage

```
hzr_argument_mapping(include_planned = TRUE)
```

Arguments

`include_planned`	Logical; if `FALSE`, only rows marked as implemented are returned.

Value

A data frame with one row per mapping rule.

See Also

`hazard()` for the target arguments, `hzr_phase()` for phase construction, and `hzr_decompos()`/`hzr_decompos_g3()` for the parameter-level early/late translations.

Examples

```
hzr_argument_mapping()
hzr_argument_mapping(include_planned = FALSE)
```

`hzr_bootstrap` *Bootstrap resampling for hazard model coefficients*

Description

Resample data with replacement, refit the hazard model on each replicate, and accumulate coefficient distributions. Returns a tidy data frame of per-replicate estimates with summary statistics. This is the R equivalent of the SAS `bootstrap.hazard.sas` macro.

Usage

```
hzr_bootstrap(
  object,
  n_boot = 200L,
  fraction = 1,
  seed = NULL,
  verbose = FALSE,
  scope = NULL,
  direction = c("both", "forward", "backward"),
  criterion = c("score", "wald", "aic"),
  slentry = 0.3,
  slstay = 0.2,
  max_steps = 50L,
  max_move = 4L,
  force_in = character(),
  force_out = character(),
  id = NULL,
  ...
)

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

Arguments

`object`	A fitted `hazard` object (with `fit = TRUE`).
`n_boot`	Integer: number of bootstrap replicates (default 200).
`fraction`	Numeric in (0, 1]: fraction of data to sample per replicate (default 1.0 for full bootstrap; < 1 for bagging).
`seed`	Optional integer random seed for reproducibility. When supplied, the call runs `set.seed(seed)` and resamples from there, so the same seed gives the same replicates. This sets the session's random number stream, as `set.seed()` would: after the call it continues from where the bootstrap's draws left it, not from where it was before. To keep a stream of your own, save `.Random.seed` before the call and restore it after, or use `withr::with_preserve_seed()`. Pass `NULL` (the default) to resample from the caller's current stream instead, which the bootstrap then advances as it draws.

verbose — Logical; if TRUE, display a text progress bar over the n_boot replicates (via utils::txtProgressBar()).

scope — **Experimental.** Candidate variable scope for embedded stepwise selection during each bootstrap replicate. The argument and the shape of the object it returns may change in a future release; see the "Selection mode is experimental" section below. NULL (default) preserves the original fixed-formula bootstrap: every replicate refits object's exact model, and summary$pct is always ~100. When supplied (a one-sided formula, character vector, or, for multiphase fits, a named list of one-sided formulas keyed by phase, matching hzr_stepwise()'s scope; a two-sided formula is an error), each replicate runs a fresh hzr_stepwise() selection instead; see Details.

direction, slentry, slstay, max_steps, max_move, force_in, force_out — Passed through to hzr_stepwise() on each replicate when scope is supplied. With scope = NULL nothing reads them, so a value other than the default is an error; the default's own value is accepted, whether or not it was passed, so that a wrapper forwarding its defaults still works. direction = "backward" with a non-empty scope is an error too: a backward screen does not read scope. For a backward screen on each replicate, pass an empty scope such as ~ 1. See hzr_stepwise() for definitions and defaults.

criterion — Entry / retention rule passed through to hzr_stepwise() on each replicate when scope is supplied; with scope = NULL, a value other than the default is an error. One of "score" (default), "wald", or "aic". "score" reproduces C/SAS HAZARD's SELECTION statistic and needs no per-candidate refit, which is what makes a bootstrap screen over many candidates tractable. Following SAS, the variance used during *selection* is approximate (shaping-parameter covariances are ignored); final-model standard errors are unaffected. For single-distribution fits, "score" computes the observed information numerically via the suggested **numDeriv** package and errors if it is not installed; a multiphase fit uses the analytic Hessian instead and does not need it. See hzr_stepwise().

id — Optional patient identifier, for data with more than one row per patient, such as time-varying covariates in counting-process form (Surv(start, stop, event) or time_lower =). Either the name of a column of the fit's data, or a vector with one value per row the fit kept. Prefer the column name: when hazard() drops rows, such as those at time 0, a vector built from the original data is the wrong length, and a column is looked up in the rows the fit kept. When given, each replicate draws patients with replacement and keeps every row of each patient drawn, so a patient drawn twice contributes all of its rows twice, and fraction counts patients rather than rows. This is SAS %HAZBOOT with _TVC=1, _IDVAR=. The default, NULL, resamples rows, which treats each row as a patient: it splits a patient's segments across replicates when they are not. A fit with entry times and no id draws a warning saying so, because the rows may not be patients; with one row per patient (plain left truncation), the warning can be ignored.

... — Additional arguments forwarded to hzr_stepwise() (e.g. control = list(maxit = 500), which every dist reads) when scope is supplied. Only control and an objective equal to the fit's are forwarded. trace is accepted and ignored, since each replicate's screen runs quietly; use verbose for progress. Any other name is an error, and so is any ... argument without scope.

x	An hzr_bootstrap object.
digits	Number of decimal places for formatting.

Details

When scope is supplied, each replicate instead runs a fresh hzr_stepwise() selection on the resampled data (starting from a fixed-shape refit of object) instead of refitting object's exact formula. This is the R equivalent of the SAS %HAZBOOT macro: fit the hazard shape with no covariates (fixing it via hzr_phase(..., fixed = "shapes")), then bootstrap-screen candidate covariates for how often they enter the model. summary$pct then reports the selection frequency across replicates, and summary$mean/sd/ci_* describe the coefficient distribution conditional on selection.

A replicate in which any fit (the base refit, a stepwise refit or the final fit) may not be a maximum, by the rule hazard() warns on with class "hzr_possible_false_maximum", is counted, and hzr_bootstrap() warns once with that count. Such replicates are kept in the pooled results. Two more kinds of replicate are counted, warned about once and kept in the same way: one whose base refit or final fit stopped on nlm() code 4 or 5 and failed the relative-gradient test, and, in selection mode, one whose screen had a candidate refit fail, so that candidate was never tested.

Value

A list with class "hzr_bootstrap" containing:

replicates Data frame with columns replicate, parameter, and estimate, one row per parameter per successful replicate.

summary Data frame with columns parameter, n, pct, mean, sd, min, max, ci_lower, ci_upper, one row per parameter. In mode = "select", pct is the selection frequency and the other statistics are conditional on selection. A free parameter whose sd is 0, to within rounding, across two or more replicates draws a warning naming it: the replicates did not re-estimate it. Parameters the fit holds fixed are exempt: those held by hzr_phase(fixed =) and shapes a constraint derives. A conserved log_mu is not exempt: it is re-solved in every replicate.

n_success Number of successfully converged replicates.

n_failed Number of replicates that failed: the refit stopped with an error, returned something other than a fit, returned a non-finite objective or one at the optimizer's -1e10 sentinel (which stands in for a likelihood that could not be evaluated), did not converge, returned a finite objective but no parameter estimates, or reported convergence on a likelihood that is flat or still rising past its estimate. A single-distribution refit that ends where the likelihood is not defined reports no objective, so it fails as a non-finite objective, not at the sentinel (#486).

failure_reasons Named integer vector counting why replicates failed, most common first: the refit's error message (or "error with an empty message"), "refit returned a <class>, not a fit object" ("an <class>" when the class begins with a vowel; or "... with no \code{fit}, not a fit object"), "refit returned no parameter estimates", or "non-finite objective (did not converge)" (the reason for a single-distribution refit that ends where the likelihood is not defined, #486), or "objective at the optimizer's -1e10 sentinel (no log-likelihood)" (a refit whose reported objective is still the sentinel), or "refit did not converge (converged = FALSE)" (a refit that reports converged = FALSE, including one stopped on a score that is not finite, #518), or "likelihood flat or still rising (no unique finite maximum)" (a refit that reports convergence where stepping its estimates

further out, along a coefficient or the direction its standard errors or Hessian say is flattest, does not lower the log-likelihood. Under separation in the resample an estimate is at infinity; with collinear columns the estimates are finite but not determined. Such replicates also draw a warning naming their count, #654), or, in select mode, `"base refit did not converge"` (a base refit with a non-finite objective) or `"base refit did not converge (converged = FALSE)"` (a base refit that reports `converged = FALSE`: the screen measures its candidates against the base, so the replicate is not screened). It sums to `n_failed`, and is an empty named integer vector, never `NULL`, when none failed. When every replicate fails, `hzr_bootstrap()` also warns, naming the most common reason.

n_uncomputable_replicates Select mode only: number of otherwise successful replicates whose screen stopped because no remaining candidate could be tested (its score statistic, or for a removal its Wald statistic, could not be computed), rather than because no candidate met `slentry` or `slstay`. A non-zero count means every reported selection frequency is biased: a candidate such a replicate could not test for entry counts as not selected, and one it could not test for removal as selected. A replicate that went on after deciding a variable without a Wald test (`wald_no_variance`) is not counted here unless it also stopped, and gets a warning of its own either way. Always `0` in refit mode.

uncomputable_reasons Select mode only: named integer vector counting *why* candidate scores were unavailable, summed over every replicate. `information_indefinite` is the one to read first: it marks candidates whose effect is too large for the score test's approximation at zero, typically strong variables. Those are refit and Wald-tested automatically, so a candidate reaching this count is one whose refit also failed and which therefore went untested, understating its selection frequency. `rows_differ` (under `criterion = "aic"`) and `loglik_below_base` (under any criterion) mark entries a replicate declined without comparing them. Such an entry counts as not selected unless a later step of the same replicate tested and entered it, so these can understate a selection frequency; a warning gives how many replicates completed after declining one. The tally counts attempts, not distinct candidates. Empty in refit mode.

n_nonmonotone_replicates Select mode only: number of otherwise successful replicates in which a forward step *lowered* the log-likelihood. Entered models are nested, so this cannot occur at the optimum; such a replicate continued from a refit that did not converge, and its later selections are pooled on the same footing as any other. Always `0` in refit mode.

n_wald_fallback_replicates, n_wald_fallbacks Select mode only: the number of otherwise successful replicates that entered at least one variable on a Wald test instead of the score statistic, and the total number of such entries across all replicates. The score criterion declines a candidate whose observed information is indefinite at `beta = 0` (which happens when the effect is *large*), so those candidates are refit and Wald-tested rather than dropped. A high count means much of the selection was decided by a different criterion from the one requested, which matters most here: these entries drive the pooled selection frequencies on the same footing as every other. Always `0` in refit mode.

mode `"refit"` (fixed-formula bootstrap) or `"select"` (embedded stepwise selection).

scope Only present when `mode == "select"`: the candidate scope used.

unresolved Only present when `mode == "select"`: the `$scope$unresolved` record of the up-front `hzr_stepwise()` screen on the original data, the names in `force_in`, `force_out` and a character `scope` that matched nothing and were ignored. `print()` shows a line when any is non-empty.

Selection mode is experimental

Everything reached through `scope` (the selection arguments, and the `summary$pct` selection frequencies they produce) is new and should be treated as unstable. The fixed-formula bootstrap (`scope = NULL`) is not affected and has been stable since 0.9.3.

Two reasons to expect change. The first is that the design is still being read off real runs rather than settled in advance: production screens have already moved the defaults once and turned up several ways a screen could report success while selecting nothing.

The second is scale, and it is the one to plan around. A screen over a large candidate pool runs for hours, and this function writes nothing until its final replicate, so a run that dies late loses everything. There is no built-in way to split one screen across processes and combine the parts. If you are running at that scale, drive `hzr_bootstrap()` in chunks from your own script and pool the replicates yourself: deriving each chunk's seed from its chunk number, offsetting replicate ids so a variable selected in two chunks is not counted once, and recomputing frequencies from the pooled replicates rather than averaging across chunks. Whatever eventually covers that inside the package may well change this function's interface.

See Also

`hazard()` for model fitting, `vcov.hazard()` for Hessian-based standard errors, `hzr_stepwise()` for the selection procedure used when `scope` is supplied.

Examples

```
data(avc)
avc <- na.omit(avc)
fit <- hazard(
  survival::Surv(int_dead, dead) ~ age + mal,
  data  = avc,
  dist  = "weibull",
  theta = c(mu = 0.01, nu = 0.5, 0, 0),
  fit   = TRUE
)
bs <- hzr_bootstrap(fit, n_boot = 50, seed = 123)
print(bs)

# Embedded stepwise selection: screen candidate covariates for how
# often they enter the model across resamples (R equivalent of SAS
# %HAZBOOT).
base <- hazard(
  survival::Surv(int_dead, dead) ~ 1,
  data  = avc,
  dist  = "weibull",
  theta = c(mu = 0.01, nu = 0.5),
  fit   = TRUE
)
bs_sel <- hzr_bootstrap(base, n_boot = 20, seed = 123,
                        scope = ~ age + mal,
                        slentry = 0.3, slstay = 0.2)
print(bs_sel)
```

`hzr_calibrate` *Calibrate a continuous variable against an outcome*

Description

Group a continuous covariate into quantile bins, compute the event probability (or hazard rate) per bin, and apply a link transform (logit, Gompertz, or Cox). This is the R equivalent of the SAS `logit.sas` and `logitgr.sas` macros.

Usage

```
hzr_calibrate(
  x,
  event,
  groups = 10L,
  by = NULL,
  link = c("logit", "gompertz", "cox"),
  time = NULL
)
```

Arguments

`x`	Numeric vector: the continuous covariate to calibrate.
`event`	Numeric vector: event indicator (1 = event, 0 = no event).
`groups`	Integer: number of quantile bins (default 10).
`by`	Optional factor or character vector for stratified calibration (SAS `logitgr.sas` functionality). If provided, calibration is computed within each stratum. Default `NULL` (no stratification).
`link`	Character: transform to apply to event probabilities. One of `"logit"` (default), `"gompertz"` (complementary log-log), or `"cox"`.
`time`	Optional numeric vector: follow-up time, required when `link = "cox"`. The Cox link computes $\log(\text{events}/\sum \text{time})$ (constant hazard rate).

Details

Use this function before model entry to assess whether the covariate relationship with the outcome is approximately linear on the link scale. If the transformed probabilities are roughly linear against the group means, the covariate can enter the model untransformed. Curvature suggests a transformation (log, quadratic) may improve fit.

Value

A data frame with one row per group (or per group-by-stratum combination) and columns:

group Integer group label.

by Stratum level (only present when by is provided).

n Number of observations in the group.

events Number of events.

mean Mean of x within the group.

min Minimum of x within the group.

max Maximum of x within the group.

prob Event probability (events / n), or for Cox link: events / sum(time).

link_value Transformed probability on the chosen link scale.

See Also

`hzr_deciles()` for model-based calibration after fitting.

Examples

```
data(avc)
avc <- na.omit(avc)

# Logit calibration of age
cal <- hzr_calibrate(avc$age, avc$dead, groups = 10)
print(cal)


if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  ggplot(cal, aes(mean, link_value)) +
    geom_point(size = 3) +
    geom_smooth(method = "lm", formula = y ~ x, se = FALSE,
                linetype = "dashed") +
    labs(x = "Age at repair (months)", y = "Logit(P(death))") +
    theme_minimal()
}
```

---

hzr_clamp_prob *Clamp probabilities away from 0 and 1*

Description

Bounds a probability vector to $[\epsilon, 1 - \epsilon]$ so that downstream $\log p$ or $\log(1 - p)$ evaluations stay finite. Used throughout the likelihood and prediction code to guard survival and CDF values against exact 0 or 1.

Usage

```
hzr_clamp_prob(p, eps = 1e-12)
```

Arguments

p	Numeric vector of probabilities.
eps	Small positive tolerance in $(0, 0.5)$; default 1e-12.

Value

Numeric vector bounded to [eps, 1 - eps].

See Also

hzr_log1pexp() and hzr_log1mexp() for the companion stable-logarithm primitives.

Examples

```
hzr_clamp_prob(c(0, 0.5, 1))
```

---

hzr_competing_risks *Competing risks cumulative incidence*

Description

Compute cumulative incidence functions for multiple competing event types using the Aalen-Johansen estimator with Greenwood variance. This is the R equivalent of the SAS markov.sas macro.

Usage

```
hzr_competing_risks(time, event)

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

Arguments

time	Numeric vector of follow-up times.
event	Integer vector of event type indicators: 0 = censored, 1 = event type 1, 2 = event type 2, etc.
x	An hzr_competing_risks object.
digits	Number of decimal places for formatting.
...	Additional arguments (ignored).

Details

Unlike the naive 1 - KM estimator (which overestimates incidence when competing risks exist), this provides the correct marginal cumulative incidence for each event type.

Value

A data frame with one row per unique event time and columns:

time Event time.

n_risk Number at risk.

n_event_1, n_event_2, ... Events of each type at this time.

n_censor Number censored at this time.

surv Overall event-free survival (freedom from all events).

incid_1, incid_2, ... Cumulative incidence for each event type.

se_surv Standard error of overall survival.

se_1, se_2, ... Standard error of each cumulative incidence.

Note

This estimator is **unweighted**: every observation contributes a unit count to the at-risk and event tallies. There is no `weights` argument, so case or inverse-probability weights are not yet supported for competing-risks incidence (unlike `hazard()`, which accepts `weights`). Pre-expand weighted rows to individual records if an approximate weighted estimate is needed.

References

Aalen O, Johansen S (1978). An empirical transition matrix for non-homogeneous Markov chains based on censored observations. *Scand J Statist* 5(3):141–150.

Kalbfleisch JD, Prentice RL (1980). *The Statistical Analysis of Failure Time Data.* Wiley, New York.

See Also

`hzr_kaplan()` for single-event survival estimation.

Examples

```
data(valves)
valves_cc <- na.omit(valves)
# Combine death and PVE into a competing risks event variable
# 0 = censored, 1 = death, 2 = PVE
event_cr <- ifelse(valves_cc$dead == 1, 1L,
                   ifelse(valves_cc$pve == 1, 2L, 0L))
time_cr <- pmin(valves_cc$int_dead, valves_cc$int_pve)
cr <- hzr_competing_risks(time_cr, event_cr)
head(cr)
```

`hzr_deciles` *Decile-of-risk calibration*

Description

Partition observations into groups (default 10) by predicted risk and compare observed vs. expected event counts in each group. Good calibration means the two track each other across the risk spectrum.

Usage

```
hzr_deciles(object, time, groups = 10L, status = NULL, event_time = NULL)
```

Arguments

`object`	A fitted `hazard` object (with `fit = TRUE`).
`time`	Numeric scalar: the horizon at which predicted survival is used to **rank subjects into risk groups** (e.g. `time = 12` ranks by 12-month predicted survival). It does not restrict the event/expected counts, which are accumulated over each subject's full follow-up.
`groups`	Integer: number of risk groups (default 10 for deciles).
`status`	Optional numeric vector of event indicators (1 = event, 0 = censored). If `NULL` (default), extracted from the fitted object's stored data.
`event_time`	Optional numeric vector of observed event/censoring times. If `NULL` (default), extracted from the fitted object. On a left-truncated row each must exceed the row's entry time, or the call stops.

Details

This reproduces the SAS `deciles.hazard.sas` macro. **All** subjects are ranked by predicted survival at the horizon `time` and split into equal-sized risk groups. Within each group the **expected** event count is the sum of each subject's predicted cumulative hazard at its *own* follow-up time, and the **observed** count is its number of events; under conservation of events the group totals sum to the total observed events. A subject with an entry time (`time_lower` on a left-truncated fit) is at risk only after it, so it contributes its cumulative hazard at exit minus its value at entry. For a weighted fit both counts carry the case weights, since such a fit conserves weighted events. Both follow `hzr_gof()`. The horizon therefore only stratifies subjects into risk groups; it does not restrict or exclude any subject, and the expected/observed totals are independent of it.

A multiphase fit that dropped the rows where a phase covariate was missing is refused with an error, as `hzr_gof()` refuses it: its design matrices hold fewer rows than there are subjects, so no subject's predicted cumulative hazard can be formed. Refit it on complete cases.

Value

A data frame with one row per risk group and columns:

group Integer group label (1 = lowest risk, ranked by predicted survival at `time`).

n Number of observations in the group.

events Observed event count in the group (all events over follow-up), weighted by the case weights for a weighted fit.

expected Expected event count: the sum of each subject's predicted cumulative hazard at its own follow-up time, minus its value at its entry time when it has one, weighted by the case weights for a weighted fit.

observed_rate Observed event rate (events / n; for a weighted fit, events per unit of case weight).

expected_rate Expected event rate (expected / n; for a weighted fit, per unit of case weight).

chi_sq Chi-square contribution: (events - expected)^2 / expected. For a weighted fit the denominator is the Poisson variance of the weighted count, the sum of each subject's squared weight times its cumulative hazard, so the statistic does not change when every weight is multiplied by the same constant.

p_value Upper-tail p-value from the chi-square test for this group (1 df).

mean_survival Mean predicted survival probability at the horizon in the group.

mean_cumhaz Mean predicted cumulative hazard at follow-up in the group: unweighted, and without the entry-time correction, so for a left-truncated or weighted fit it is not `expected / n`.

An attribute `"overall"` is attached with the overall chi-square statistic, degrees of freedom, and p-value.

See Also

`predict.hazard()` for the prediction types used internally.

Examples

```
data(avc)
avc <- na.omit(avc)
fit <- hazard(
  survival::Surv(int_dead, dead) ~ age + mal,
  data  = avc,
  dist  = "weibull",
  theta = c(mu = 0.01, nu = 0.5, beta_age = 0, beta_mal = 0),
  fit   = TRUE
)
cal <- hzr_deciles(fit, time = 120)
print(cal)
```

`hzr_decompos` *Generalized temporal decomposition*

Description

Computes the cumulative distribution $G(t)$, density $g(t)$, and hazard $h(t) = g(t)/(1 - G(t))$ for the parametric family defined by half-life, time exponent, and shape. It supplies the `"cdf"` and `"hazard"` phase shapes of a multiphase hazard model; the `"g3"` late phase comes from `hzr_decompos_g3()`, and the `"constant"` phase is linear in time.

Usage

```
hzr_decompos(time, t_half, nu, m)
```

Arguments

`time`	Numeric vector of times (must be > 0).
`t_half`	Half-life: time at which $G(t_{1/2}) = 0.5$. Must be > 0.
`nu`	Time exponent controlling rate dynamics. SAS early: `NU`. SAS late: relates to `GAMMA/ETA`.
`m`	Shape exponent controlling the distributional form. SAS early: `M`. SAS late: relates to `GAMMA/ALPHA`.

Value

A named list with four numeric vectors, each the same length as `time`:

G Cumulative distribution $G(t) \in [0, 1]$.

g Density $g(t) = dG/dt \geq 0$. The "early" phase temporal pattern.

h Hazard $h(t) = g(t)/(1 - G(t)) \geq 0$. The "late" phase temporal pattern.

log_surv $\log(1 - G(t))$, computed from each case's own log-scale terms rather than from `G`, so it keeps its accuracy where `G` rounds to 1. `-log_surv` is the cumulative hazard of a `"hazard"` phase, and `h` is computed as $\exp(\log g - \log(1 - G))$.

Parameter mapping from SAS/C HAZARD

The original C code used separate parameterizations for early (DELTA, RHO/THALF, NU, M) and late (TAU, GAMMA, ALPHA, ETA) phases. The early phase maps onto the three parameters here: DELTA must be 0, and RHO is fixed by THALF, NU and M. The late phase does not: it is a separate four-parameter shape computed by `hzr_decompos_g3()`. See `hzr_argument_mapping()` for the full translation table.

Valid parameter combinations

Six cases are defined by the signs of nu and m:

Case	Sign	Behavior
1	m > 0, nu > 0	Standard sigmoidal
1L	m = 0, nu > 0	Exponential-like (Weibull CDF)
2	m < 0, nu > 0	Heavy-tailed
2L	m < 0, nu = 0	Exponential decay
3	m > 0, nu < 0	Bounded cumulative
3L	m = 0, nu < 0	Bounded exponential

The combination m < 0 **and** nu < 0 is undefined and raises an error. `nu = 0` is supported only with m < 0 (Case 2L, the exponential-decay limit); `nu = 0` with m >= 0 has no usable limiting form and raises an error.

Mathematical form

The construction fixes a rate ρ so that $G(t_{1/2}) = 0.5$ exactly. For the base case ($m > 0,\ \nu > 0$):

$$\rho = \nu\, t_{1/2} \left(\frac{2^m - 1}{m} \right)^{\nu}, \qquad b(t) = \frac{\nu t}{\rho}$$

The CDF and density are then

$$G(t) = \left(1 + m\, b(t)^{-1/\nu}\right)^{-1/m}, \qquad g(t) = \left(1 + m\, b(t)^{-1/\nu}\right)^{-1/m-1} b(t)^{-1/\nu-1}/\rho$$

and the hazard is $h(t) = g(t)/(1-G(t))$. The remaining five cases in the table above arise as limits ($m \to 0, \nu \to 0$) or sign reflections of this base form; the implementation dispatches to the appropriate branch after inspecting the signs of nu and m. See `vignette("mf-mathematical-foundations")` for the full derivation of every case.

References

Blackstone EH, Naftel DC, Turner ME Jr. The decomposition of time-varying hazard into phases, each incorporating a separate stream of concomitant information. *J Am Stat Assoc.* 1986;81(395):615–624. doi:10.1080/01621459.1986.10478314

Rajeswaran J, Blackstone EH, Ehrlinger J, Li L, Ishwaran H, Parides MK. Probability of atrial fibrillation after ablation: Using a parametric nonlinear temporal decomposition mixed effects model. *Stat Methods Med Res.* 2018;27(1):126–141. doi:10.1177/0962280215623583

See Also

`hzr_phase_cumhaz()` for the phase-level cumulative hazard contribution, `hzr_argument_mapping()` for SAS/C parameter mapping, `hzr_phase()` for specifying phases in `hazard()` models.

`vignette("mf-mathematical-foundations")` for the full derivation.

Examples

```
t_grid <- seq(0.1, 10, by = 0.1)

# Case 1: standard sigmoidal (m > 0, nu > 0)
d1 <- hzr_decompos(t_grid, t_half = 3, nu = 2, m = 1)
plot(t_grid, d1$G, type = "l", main = "CDF (m=1, nu=2)")

# Case 1L: Weibull-like (m = 0, nu > 0)
d1L <- hzr_decompos(t_grid, t_half = 3, nu = 2, m = 0)

# Case 2: heavy-tailed (m < 0, nu > 0)
d2 <- hzr_decompos(t_grid, t_half = 3, nu = 2, m = -1)
```

---

hzr_decompos_g3 *Late-phase (G3) temporal decomposition*

Description

Computes the cumulative intensity $G_3(t)$ and its derivative $g_3(t) = dG_3/dt$ for the late-phase parametric family used in the original Blackstone C/SAS HAZARD code. Unlike `hzr_decompos()` (which computes the early-phase G1, a bounded CDF), this function can produce **unbounded** values, making it suitable for modelling increasing late risk.

Usage

```
hzr_decompos_g3(time, tau, gamma, alpha, eta)
```

Arguments

`time`	Numeric vector of times (must be > 0).
`tau`	Positive scalar scale parameter.
`gamma`	Positive scalar time exponent.
`alpha`	Non-negative scalar shape parameter (0 selects limiting case).
`eta`	Positive scalar outer exponent.

Value

A named list with two numeric vectors, each the same length as `time`:

G3 Cumulative intensity $G_3(t) \geq 0$ (may exceed 1).

g3 Derivative $g_3(t) = dG_3/dt \geq 0$.

Mathematical form

When $\alpha > 0$:

$$G_3(t) = \left(\left((t/\tau)^\gamma + 1\right)^{1/\alpha} - 1\right)^\eta$$

When $\alpha = 0$ (limiting exponential case):

$$G_3(t) = \left(\exp\left((t/\tau)^\gamma\right) - 1\right)^\eta$$

Parameter mapping from SAS/C HAZARD

SAS name	R argument	Role
TAU	`tau`	Scale (time at which $(t/\tau) = 1$)
GAMMA	`gamma`	Power exponent on t/τ
ALPHA	`alpha`	Shape (0 = exponential limiting case)
ETA	`eta`	Outer power exponent

References

Blackstone EH, Naftel DC, Turner ME Jr. The decomposition of time-varying hazard into phases, each incorporating a separate stream of concomitant information. *J Am Stat Assoc.* 1986;81(395):615–624. doi:10.1080/01621459.1986.10478314

See Also

`hzr_decompos()` for the early-phase (G1) decomposition, `hzr_phase_cumhaz()` for phase-level cumulative hazard helpers.

Examples

```
t_grid <- seq(0.1, 10, by = 0.1)

# Weibull-like: alpha = 1 gives G3(t) = (t/tau)^(gamma*eta)
d <- hzr_decompos_g3(t_grid, tau = 1, gamma = 3, alpha = 1, eta = 1)
plot(t_grid, d$G3, type = "l", main = "G3: power law (gamma=3)")

# General case with alpha > 0
d2 <- hzr_decompos_g3(t_grid, tau = 2, gamma = 2, alpha = 0.5, eta = 1)
```

`hzr_evaluate` *Evaluate a hazard model at parameters you supply*

Description

Computes a model's objective, and optionally its hazard and cumulative hazard, at parameters **you** supply rather than at parameters fitted from the data. The objective is the log-likelihood, except for a model built with `objective = "sas"` on data with interval-censored rows, where it is PROC HAZARD's interval-mean-hazard objective (see `objective` under Value). This is what a parity check needs: the objective at another program's converged estimates, evaluated by this package's own code.

Usage

```
hzr_evaluate(object, theta, times = NULL)
```

Arguments

`object`	A `hazard` object, fitted or built with `fit = FALSE`. Its data, distribution and phase specification are used; its own `theta` is not.
`theta`	Numeric vector of parameters to evaluate at, on the internal scale the model uses (see `hzr_theta_names()`). Its length must match the model's parameter count. Names, when supplied, must match too.
`times`	Optional numeric vector of times, for multiphase models only. When given, the hazard and cumulative hazard at those times are returned for a covariate-free ("baseline") subject: every covariate at 0, so the curve is the shape the supplied parameters describe, not a prediction for any row of the data. The other families have no internal shape function that takes parameters directly, and writing their hazards out here would duplicate `predict()`; `times` is refused for them rather than mirrored.

Details

The result is not a fit and does not pretend to be one. It carries no standard errors, no convergence status and no covariance matrix, because nothing was estimated: `print()` says so on its first line, and the parameters are labelled as supplied. Where a fitted model is what you want, use `hazard(..., fit = TRUE)`.

A phase built with `hzr_phase(constraint = )` derives one of its shapes from the others, and that rule is applied to `theta` here as the fit applies it, so the derived entry you pass is replaced rather than used as given. At a fitted model's own estimates this returns that fit's objective, under Conservation of Events too: since #362 the fit's objective is recomputed at the estimates it returns, except where the fit warns that it could not, and then the two can differ.

A `theta` that passes the input checks but that the likelihood cannot evaluate – an overflowing rate, a shape outside the family, or a value past a guard that stops short of where the log-likelihood itself overflows – gives `-Inf`, with a warning of class `"hzr_evaluate_not_finite"`, for every distribution. A Weibull `mu` or `nu` at or below 0 is still refused outright by those input checks.

Value

An object of class `hzr_evaluation`: a list with `theta`, the parameters the likelihood was evaluated at – the vector supplied, with any constrained entry replaced by the value its phase derives, which is warned about as `hazard()` warns; `logLik`, the log-likelihood there, or, for a model built with `objective = "sas"`, that objective, which is not a log-likelihood wherever the data have interval-censored rows; `objective`, `"likelihood"` or `"sas"`, saying which of the two `logLik` holds; `dist`; `n_obs` and `n_events`, the rows the likelihood scored and the exact events among them (a left- or interval-censored row counts in `n_obs`, not in `n_events`); and, when `times` was given, `curve`, a data frame of `time`, `hazard` and `cumulative_hazard`, which is `NA` throughout where `logLik` is `-Inf`.

See Also

`hazard()` to fit a model, `hzr_theta_names()` for the parameter order.

Examples

```
data(avc, package = "TemporalHazard")
spec <- hazard(survival::Surv(int_dead, dead) ~ 1, data = avc,
               dist = "weibull", theta = c(1, 1), fit = FALSE)
# The likelihood at parameters from somewhere else, e.g. a SAS run.
hzr_evaluate(spec, theta = c(0.05, 0.9))
```

---

hzr_gof *Goodness-of-fit: observed vs. predicted events*

Description

Compare a fitted hazard model with the data two ways: its survival curve against the nonparametric Kaplan-Meier estimate, and the number of events it expects against the number observed, tallied over follow-up. This is the R equivalent of the SAS `hazplot.sas` macro and implements the conservation-of-events diagnostic.

Usage

```
hzr_gof(object, time_grid = NULL)
```

Arguments

`object`	A fitted `hazard` object (with `fit = TRUE`).
`time_grid`	Optional numeric vector of time points at which to evaluate the parametric model. If `NULL` (default), uses the distinct Kaplan-Meier times of the fitted data: the event and censoring times, with any closer together than `survival::survfit()`'s tolerance merged into one. A supplied grid must hold finite, non-negative times. It is sorted and exact repeats dropped, because the cumulative columns accumulate in time order.

Details

The diagnostic is for right-censored data: every stored status must be 0 (censored) or 1 (event). A fit with any left-censored (status -1) or interval-censored (status 2) row is refused with an error. So is a multiphase fit that dropped the rows where a covariate was missing: its design matrices then hold fewer rows than there are patients, so no patient-by-patient tally can be formed. Refit it on complete cases.

At each time point the function computes:

- The Kaplan-Meier survival and cumulative hazard.
- The parametric survival and cumulative hazard from the fitted model at the covariate means (and per-phase components for multiphase models). This is the curve to plot against the Kaplan-Meier estimate.
- Cumulative observed events vs. cumulative expected events. Each patient's expected count is their own cumulative hazard, from their own covariates, at the end of their follow-up, less their cumulative hazard at entry when the fit is left truncated (`time_lower` on a status 0 or 1 row). These are summed over the patients leaving follow-up at each time.
- The running residual (expected minus observed).

The conservation-of-events principle says a model fit by maximum likelihood predicts as many events as were observed: add up every patient's cumulative hazard over their follow-up and you get the event count back. The final residual is then zero and the printed "Conservation ratio (E/O)" is 1. A multiphase fit with Conservation of Events applied (`control = list(conserve = TRUE)`, the default) meets the identity by construction. Weibull and exponential fits meet it at the exact maximum, so at a converged fit E/O sits close to 1, off only by how far short of the maximum the optimizer stopped. The log-logistic and log-normal models carry no such identity, and for them E/O is a check of calibration in total.

The parametric curve is a different quantity. For a model with covariates it belongs to one "mean patient" with average covariates, and the mean patient's cumulative hazard is not the average of the patients' cumulative hazards, so `par_cumhaz` does not enter the expected count. For an intercept-only model every patient shares that curve. Without entry times each patient's expected count is the curve at their exit time; with entry times it is the curve's rise from entry to exit, so the two differ. The means are those of the design-matrix columns, taken phase by phase for a multiphase fit, whether its covariates enter globally, through the phase formulas or both, so a factor enters as the proportion of patients in each level. With `time_windows`, a phase built on the global covariates carries their means in the window that contains each time; a phase formula's own columns keep their plain means.

For a weighted fit both tallies carry the case weights: observed events are $\sum_i w_i d_i$ and expected events $\sum_i w_i H_i$, the form in which a weighted fit conserves events. The `n_risk`, `n_event`, `n_censor` and Kaplan-Meier columns are unweighted.

Each patient is placed at the Kaplan-Meier time `survival::survfit()` gives them, which merges exit times closer together than its tolerance. On the default grid every patient lands on a grid point, and `hzr_gof()` warns if one cannot be placed. With a custom `time_grid`, a patient is counted in both tallies only if that time, or failing it their own follow-up time, falls on a grid point. A grid such as `seq()` over the follow-up holds few exit times, so `hzr_gof()` warns with the number left out, and the totals, residual and E/O then describe only the patients counted. Include the exit times in `time_grid` to count everyone.

Value

A data frame with one row per time point and columns:

time Evaluation time.

n_risk Number at risk at this time: subjects still in follow-up. A subject with an entry time is at risk only after it.

n_event Number of events at this time.

n_censor Number censored at this time.

km_surv Kaplan-Meier survival estimate, using the counting-process risk set when the fit has entry times.

km_cumhaz Kaplan-Meier cumulative hazard ($-\log(\text{km_surv})$).

par_surv Parametric survival from the fitted model, at the covariate means for a model with covariates. For plotting against `km_surv`; not used for `cum_expected`.

par_cumhaz Parametric cumulative hazard, at the covariate means for a model with covariates. For plotting; not used for `cum_expected`.

cum_observed Cumulative observed events to this time, weighted by the case weights for a weighted fit. With a custom `time_grid`, only the patients whose exit is a grid point up to this time are counted (see Details).

cum_expected Cumulative expected events: over the patients leaving follow-up by this time (with a custom `time_grid`, the same patients as `cum_observed`), the sum of each patient's own cumulative hazard at exit minus that at entry, weighted by the case weights for a weighted fit. With `time_windows`, both cumulative hazards use the patient's covariate window at exit, as the likelihood does.

residual Expected minus observed (`cum_expected - cum_observed`).

For multiphase models, additional columns are appended for each phase: `par_cumhaz_<phase>`, also at the covariate means.

An attribute `"summary"` is attached with scalar diagnostics: total observed events, total expected events, the final residual, and `n_tallied`, the number of patients the two tallies cover, out of `n`.

See Also

`hzr_deciles()` for decile-of-risk calibration, `predict.hazard()` for prediction types.

Examples

```
data(avc)
avc <- na.omit(avc)
fit <- hazard(
  survival::Surv(int_dead, dead) ~ age + mal,
  data  = avc,
  dist  = "weibull",
  theta = c(mu = 0.01, nu = 0.5, beta_age = 0, beta_mal = 0),
  fit   = TRUE
)
gof <- hzr_gof(fit)
print(gof)
```

```
# Expected events are summed per patient.  This fit has no entry times,
# so the total is the sum of each patient's own cumulative hazard at
# their follow-up time:
nd <- avc[, c("age", "mal")]
nd$time <- avc$int_dead
c(hzr_gof = attr(gof, "summary")$total_expected,
  predict = sum(predict(fit, newdata = nd, type = "cumulative_hazard")))

# Plot observed vs expected events
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  ggplot(gof, aes(x = time)) +
    geom_line(aes(y = cum_observed), colour = "#D55E00") +
    geom_line(aes(y = cum_expected), colour = "#0072B2") +
    labs(x = "Time", y = "Cumulative events") +
    theme_minimal()
}
```

`hzr_kaplan` *Kaplan-Meier survival with exact logit confidence limits*

Description

Compute the product-limit (Kaplan-Meier) survival estimate with logit-transformed confidence limits that respect the $[0, 1]$ boundary. This is the R equivalent of the SAS `kaplan.sas` macro.

Usage

```
hzr_kaplan(time, status, conf_level = 0.95, event_only = TRUE)
```

Arguments

`time`	Numeric vector of follow-up times.
`status`	Numeric event indicator (1 = event, 0 = censored).
`conf_level`	Confidence level for the interval (default 0.95). The SAS default of 0.68268948 corresponds to a 1-SD interval.
`event_only`	Logical; if `TRUE` (default), only return rows at event times (where `n_event > 0`). If `FALSE`, return rows at all times reported by `survival::survfit()` (events and censoring times).

Details

The standard Greenwood confidence interval can exceed $[0, 1]$ in the tails. The logit-transformed interval avoids this by working on the log-odds scale:

$$\mathrm{CL}_{\text{lower}} = S / \big(S + (1-S)\,\exp(z_\alpha\,\mathrm{SI})\big)$$
$$\mathrm{CL}_{\text{upper}} = S / \big(S + (1-S)\,\exp(-z_\alpha\,\mathrm{SI})\big)$$

where $\mathrm{SI} = \sqrt{V_P - 1}/(1-S)$, V_P is the cumulative Greenwood variance product, and z_α is the normal quantile for the requested confidence level.

Value

A data frame with one row per time point and columns:

time Event/censoring time.

n_risk Number at risk at start of interval.

n_event Number of events at this time.

n_censor Number censored at this time.

survival Kaplan-Meier survival estimate.

std_err Standard error of survival (Greenwood).

cl_lower Lower confidence limit (logit-transformed).

cl_upper Upper confidence limit (logit-transformed).

cumhaz Cumulative hazard $= -\log(S)$.

hazard Interval hazard rate $= \log(S_{t-1}/S_t)/\Delta t$.

density Probability density estimate $= (S_{t-1} - S_t)/\Delta t$.

life Restricted mean survival time (area under curve to this time).

References

Kaplan EL, Meier P (1958). Nonparametric estimation from incomplete observations. *J Am Stat Assoc* 53(282):457–481. doi:10.1080/01621459.1958.10501452

Greenwood M (1926). The natural duration of cancer. *Reports on Public Health and Medical Subjects* 33:1–26.

See Also

`hzr_gof()` for parametric vs. nonparametric comparison.

Examples

```
data(cabgkul)
km <- hzr_kaplan(cabgkul$int_dead, cabgkul$dead)
head(km)


if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)
  ggplot(km, aes(time)) +
    geom_step(aes(y = survival * 100)) +
```

```
    geom_ribbon(aes(ymin = cl_lower * 100, ymax = cl_upper * 100),
                stat = "identity", alpha = 0.2) +
    labs(x = "Months", y = "Survival (%)") +
    theme_minimal()
}
```

`hzr_log1mexp` *Numerically stable log(1 - exp(-x)) for x > 0*

Description

Computes

$$\mathrm{log1mexp}(x) = \log\bigl(1 - e^{-x}\bigr), \qquad x > 0$$

accurately across the full range. Following Mächler (2012), the evaluation switches at $x = \log 2$: for small x it uses $\log(-\mathrm{expm1}(-x))$ (avoiding cancellation as $x \to 0$), and for larger x it uses $\mathrm{log1p}(-e^{-x})$ (avoiding loss of precision as $x \to \infty$). Non-positive or non-finite inputs return `NA`.

Usage

```
hzr_log1mexp(x)
```

Arguments

`x`	Numeric vector with positive values.

Value

Numeric vector with element-wise $\log(1 - e^{-x})$.

References

Mächler M (2012). *Accurately Computing* $\log(1 - e^{-|a|})$. R package Rmpfr vignette. https://CRAN.R-project.org/package=Rmpfr/vignettes/log1mexp-note.pdf

See Also

`hzr_log1pexp()` for the softplus $\log(1 + e^{x})$, `hzr_clamp_prob()` for boundary-safe probabilities.

Examples

```
hzr_log1mexp(c(0.01, 0.5, 5))
```

`hzr_log1pexp` *Numerically stable log(1 + exp(x))*

Description

Computes the "softplus" function

$$\mathrm{log1pexp}(x) = \log\bigl(1 + e^{x}\bigr)$$

without overflow. A naive `log(1 + exp(x))` overflows to `Inf` for $x \gtrsim 710$ even though the true value is $\approx x$. The identity $\log(1 + e^{x}) = x + \log(1 + e^{-x})$ for $x > 0$ keeps every intermediate finite.

Usage

```
hzr_log1pexp(x)
```

Arguments

`x`	Numeric vector.

Value

Numeric vector with element-wise $\log(1 + e^{x})$.

References

Mächler M (2012). *Accurately Computing* $\log(1 - e^{-|a|})$. R package Rmpfr vignette. https://CRAN.R-project.org/package=Rmpfr/vignettes/log1mexp-note.pdf

See Also

`hzr_log1mexp()` for the complementary $\log(1-e^{-x})$, `hzr_clamp_prob()` for boundary-safe probabilities.

Examples

```
hzr_log1pexp(c(-50, 0, 50))
```

`hzr_nelson` *Wayne Nelson cumulative hazard estimator with lognormal confidence limits*

Description

Compute the Nelson-Aalen cumulative hazard estimate with lognormal confidence limits. Supports weighted events for severity-adjusted analyses of repeated/recurrent events. This is the R equivalent of the SAS `nelsonl.sas` macro.

Usage

```
hzr_nelson(time, event, weight = NULL, conf_level = 0.95)

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

Arguments

`time`	Numeric vector of follow-up times.
`event`	Numeric event indicator (1 = event, 0 = censored).
`weight`	Optional numeric vector of event weights (default 1). Weights are applied only to events (censored observations contribute zero weight). Use for severity-weighted repeated events.
`conf_level`	Confidence level for the interval (default 0.95). SAS `%NELSONT` defaults to `CLEVEL = 0.68268948`, a 1-SD (68.3%) interval, so a band reproduced at this function's default is about 1.96 times wider than the SAS one. Pass `conf_level = 2 * stats::pnorm(1) - 1` to match. Same convention as `hzr_kaplan()`.
`x`	An `hzr_nelson` object.
`digits`	Number of decimal places for formatting.
`...`	Additional arguments (ignored).

Details

Unlike `survival::survfit()` which uses the Breslow estimator with Greenwood variance, this function uses the Wayne Nelson estimator with lognormal confidence limits that are always non-negative.

Value

A data frame with one row per unique event time and columns:

time Event time.

n_risk Number at risk.

n_event Number of events at this time.

weight_sum Sum of event weights at this time.

cumhaz Nelson cumulative hazard estimate.

std_err Standard error.

cl_lower Lower lognormal confidence limit.

cl_upper Upper lognormal confidence limit.

hazard Interval hazard rate.

cum_events Cumulative (weighted) event count.

References

Nelson W (1972). Theory and applications of hazard plotting for censored failure data. *Technometrics* 14(4):945–966. doi:10.1080/00401706.1972.10488991

Aalen O (1978). Nonparametric inference for a family of counting processes. *Ann Statist* 6(4):701–726. doi:10.1214/aos/1176344247

See Also

`hzr_kaplan()` for survival estimation.

Examples

```
data(cabgkul)
nel <- hzr_nelson(cabgkul$int_dead, cabgkul$dead)
head(nel)
```

`hzr_phase` *Specify a single hazard phase*

Description

Creates an `hzr_phase` object describing one term in a multiphase additive cumulative hazard model. Pass a list of these to the `phases` argument of `hazard()` when `dist = "multiphase"`.

Usage

```
hzr_phase(
  type = c("cdf", "hazard", "constant", "g3"),
  t_half = 1,
  nu = 1,
  m = 0,
  tau = 1,
  gamma = 1,
  alpha = 1,
  eta = 1,
  formula = NULL,
```

```
  fixed = character(0),
  constraint = c("none", "alpha_gamma_eta", "eta_gamma")
)

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

Arguments

type	Character; the phase's temporal shape, one of "cdf" (early resolving risk), "hazard" (accumulating G1 aging risk), "g3" (late rising risk, the original C/SAS late phase), or "constant" (flat background rate). See the **Phase types** section for what each means and when to use it.
t_half	Positive scalar; initial half-life (time at which $G(t_{1/2}) = 0.5$). Used for "cdf" and "hazard" phases. SAS early: THALF/RHO.
nu	Numeric scalar; initial time exponent. Used for "cdf" and "hazard" phases. SAS early: NU.
m	Numeric scalar; initial shape exponent. Used for "cdf" and "hazard" phases. SAS early: M.
tau	Positive finite scalar; scale parameter for "g3" phases. SAS late: TAU.
gamma	Positive finite scalar; time exponent for "g3" phases. SAS late: GAMMA.
alpha	Non-negative scalar; shape parameter for "g3" phases. When alpha > 0, the generic G3 formula is used; alpha = 0 gives the exponential limiting case. SAS late: ALPHA.
eta	Positive finite scalar; outer exponent for "g3" phases. SAS late: ETA.
formula	Optional one-sided formula (e.g. ~ age + nyha) for phase-specific covariates. It is evaluated in the data given to hazard(), so without data hazard() refuses it, unless it builds nothing either way: an intercept-only ~ 1 with no global x. The phase's scale parameter plays the role of an intercept, so the design never has one: removing it (~ 0 + age, ~ age - 1) is ignored with a warning, and builds the design of ~ age. When NULL (default), the phase inherits the global design from hazard(): the global formula's covariates, or x on the vector interface.
fixed	Character vector naming shape parameters to hold fixed during optimization. Valid names for "cdf"/"hazard": "t_half", "nu", "m", or "shapes" (shorthand for all three). Valid names for "g3": "tau", "gamma", "alpha", "eta", or "shapes" (shorthand for all four). Fixed parameters are held at their starting values; only mu (and covariates) are estimated. Ignored for "constant" phases. This mirrors the SAS/C HAZARD workflow where shapes are typically fixed and only scale parameters are estimated.
constraint	For "g3" phases, a rule that *derives* one shape from the others rather than estimating it: "none" (default) every shape is estimated or fixed. "alpha_gamma_eta" $\alpha = \gamma\eta/2$, so that $\gamma\eta/\alpha = 2$. SAS/C: FIXGAE2. "eta_gamma" $\eta = 2/\gamma$, so that $\gamma\eta = 2$. SAS/C: FIXGE2.

At alpha = 1, hzr_phase()'s default, the g3 form is $(t/\tau)^{\gamma\eta}$ (see hzr_decompos_g3()), which depends on γ and η only through their product. So γ and η are not separately identified there: under "eta_gamma" the product is fixed at 2 and γ is not identified at all, and with both estimated only the product is. With alpha *fixed* at 1, hazard() therefore fits the phase as PROC HAZARD does: tau is held at 1 and the product is carried by one parameter (under "eta_gamma", $\gamma = 2$ and $\eta = 1$ are both held), with a warning and a record in fitfitboundary. Under "eta_gamma" a fixed alpha above 1 is refused, as PROC HAZARD refuses it, and a free one started at 1 or above starts at 2/3 instead. A free alpha that comes to rest near 1 can still leave γ undetermined; the weak-direction warning names it when the Hessian shows it. The derived parameter follows the others at every step of the optimization, so it is not a free parameter and cannot be named in fixed; "shapes" leaves it out. Its starting value is computed from the others, and a value you supply for it, here or in the theta given to hazard(), is replaced, with a warning when it differs. Under hazard(fit = FALSE) that replacement is made only when no phase carries covariates, since otherwise its slot in theta is not known until the design is built; hazard() warns when it could not be made. Its standard error in vcov() is the delta-method one, carried from the parameters it is derived from, so confidence limits from predict() include its uncertainty. Only one constraint per phase: SAS's FIXGE2 and FIXGAE2 together force $\alpha = 1$ and fix tau, gamma and eta, which is better written as those fixed values.

x — An hzr_phase object (for print.hzr_phase()).

... — Additional arguments (ignored).

Value

An S3 object of class "hzr_phase" with elements:

type Phase type string.

t_half Initial half-life (cdf/hazard phases).

nu Initial time exponent (cdf/hazard phases).

m Initial shape exponent (cdf/hazard phases).

tau Scale parameter (g3 phases).

gamma Time exponent (g3 phases).

alpha Shape parameter (g3 phases).

eta Outer exponent (g3 phases).

formula Phase-specific formula or NULL.

fixed Character vector of fixed parameter names (may be empty).

constraint The shape constraint (g3 phases).

Role in the multiphase model

Each phase is one term j in the additive cumulative hazard

$$H(t \mid \mathbf{x}) = \sum_{j=1}^{J} \mu_j(\mathbf{x})\, \Phi_j(t)$$

where $\mu_j(\mathbf{x}) = \exp(\alpha_j + \mathbf{x}_j^\top \beta_j)$ is the phase-specific log-linear scale and $\Phi_j(t)$ is the temporal shape selected by `type` (below). The `t_half`/`nu`/`m` (or g3 `tau`/`gamma`/`alpha`/`eta`) arguments set the starting values for that shape; `formula` attaches the covariates $\mathbf{x}_j$ that enter μ_j.

Phase types

The `type` argument chooses the temporal shape $\Phi_j(t)$ for the phase. Each captures a qualitatively different pattern of risk over time; a typical clinical model combines an *early*, a *constant*, and a *late* phase so that the total hazard can fall, level off, and rise again.

`"cdf"`**: early, resolving risk** Named for the **c**umulative **d**istribution **f**unction: the phase contributes $\Phi(t) = G(t)$, the bounded CDF of the temporal decomposition (0 at $t = 0$, rising to a ceiling of 1). Because it saturates, the *hazard* it adds, $\mu\, g(t)$, peaks early and then decays toward zero (the signature of a one-time insult that patients either succumb to or survive past, e.g. peri-operative mortality). Shape set by `t_half`, `nu`, `m`. SAS/C equivalent: the Early (G1) phase.

`"hazard"`**: accumulating aging risk (G1 family)** Named because the phase contributes a **cumulative hazard** built from the same G1 family: $\Phi(t) = -\log(1 - G(t))$, which is unbounded and monotone increasing. Its hazard $\mu\, h(t)$ rises without leveling off, so it models risk that grows as subjects age. This is an alternative late-risk form derived from G1; for the original SAS/C late phase prefer `"g3"`. Shape set by `t_half`, `nu`, `m`.

`"g3"`**: late, rising risk (original C/SAS late phase)** Named for the **G3** (third) decomposition family used by the original HAZARD program for the late phase. It contributes $\Phi(t) = G_3(t)$ from `hzr_decompos_g3()`, an unbounded intensity with its own four-parameter shape (`tau`, `gamma`, `alpha`, `eta`) that is more flexible than the G1-derived `"hazard"` form for capturing accelerating late mortality (e.g. structural valve deterioration years after surgery). Use this when reproducing classic three-phase HAZARD models. SAS/C equivalent: the Late (G3) phase.

`"constant"`**: flat background rate** A time-invariant hazard: $\Phi(t) = t$, so the added hazard μ is constant (the exponential model). It represents the steady, ongoing risk present at all follow-up times, independent of how long ago the time origin was. Takes no shape parameters; only its scale μ (and any covariates) is estimated. SAS/C equivalent: the Constant (G2) phase.

The shape derivative $\varphi_j = d\Phi_j/dt$ (which forms the instantaneous hazard contribution $\mu_j\, \varphi_j(t)$) is $g(t)$ for `"cdf"`, $h(t)$ for `"hazard"`, $g_3(t)$ for `"g3"`, and 1 for `"constant"`.

See Also

`hazard()` for fitting multiphase models, `hzr_decompos()` for the underlying parametric family, `hzr_phase_cumhaz()` and `hzr_phase_hazard()` for computing $\Phi(t)$ and $\phi(t)$ from these specifications.

`vignette("fitting-hazard-models")` for multiphase fitting examples, `vignette("mf-mathematical-foundations")` for the mathematical framework.

Examples

```
# Classic 3-phase Blackstone pattern
early <- hzr_phase("cdf",      t_half = 0.5, nu = 2, m = 0)
const <- hzr_phase("constant")
late  <- hzr_phase("g3", tau = 1, gamma = 3, alpha = 1, eta = 1)

# Fix all shapes (C/SAS-style: only estimate mu)
early_fixed <- hzr_phase("cdf", t_half = 0.5, nu = 2, m = 0,
                         fixed = "shapes")
late_fixed  <- hzr_phase("g3", tau = 1, gamma = 3, alpha = 1, eta = 1,
                         fixed = "shapes")

# Derive alpha from gamma and eta (SAS/C FIXGAE2)
late_gae2 <- hzr_phase("g3", tau = 14, gamma = 22, eta = 0.18,
                       constraint = "alpha_gamma_eta")

# Fix only some parameters
early_partial <- hzr_phase("cdf", t_half = 0.5, nu = 2, m = 0,
                           fixed = c("nu", "m"))

# Phase with specific covariates
early_cov <- hzr_phase("cdf", t_half = 0.5, nu = 2, m = 0,
                       formula = ~ age + shock)

# Use in hazard():
# hazard(Surv(time, status) ~ age, data = dat,
#        dist = "multiphase",
#        phases = list(early = early, constant = const, late = late))
```

---

hzr_phase_cumhaz *Cumulative hazard contribution from a single phase*

Description

Computes $\Phi_j(t)$ for one phase in the additive model $H(t|x) = \sum_j \mu_j(x)\Phi_j(t)$.

Usage

```
hzr_phase_cumhaz(
  time,
  t_half = 1,
  nu = 1,
  m = 0,
  type = c("cdf", "hazard", "constant")
)
```

Arguments

`time`	Numeric vector of times (> 0).
`t_half`	Half-life parameter (> 0).
`nu`	Time exponent.
`m`	Shape parameter.
`type`	Phase type: `"cdf"` (early, uses $G(t)$), `"hazard"` (late, uses cumulative hazard from $h(t)$), or `"constant"` (flat rate, $\Phi = t$).

Details

- `"cdf"`: $\Phi(t) = G(t)$. Bounded $[0, 1]$. Models early risk that resolves over time.
- `"hazard"`: $\Phi(t) = -\log(1 - G(t))$. Monotone increasing. Models late or aging risk. This is the cumulative hazard derived from the hazard function $h(t)$, since $\int_0^t h(s)\, ds = -\log(1 - G(t))$.
- `"constant"`: $\Phi(t) = t$. Ignores `t_half`, `nu`, `m`. Equivalent to exponential (constant hazard rate).

Value

Numeric vector of cumulative hazard contributions $\Phi(t)$, same length as `time`.

See Also

`hzr_decompos()` for the underlying parametric family, `hzr_phase_hazard()` for the instantaneous hazard contribution.

Examples

```
t_grid <- seq(0.1, 10, by = 0.1)
phi_early <- hzr_phase_cumhaz(t_grid, t_half = 2, nu = 2, m = 0,
                              type = "cdf")
phi_late  <- hzr_phase_cumhaz(t_grid, t_half = 5, nu = 1, m = 0,
                              type = "hazard")
phi_const <- hzr_phase_cumhaz(t_grid, type = "constant")
```

---

hzr_phase_hazard *Instantaneous hazard contribution from a single phase*

Description

Computes $\phi_j(t) = d\Phi_j/dt$ for one phase, the derivative of the cumulative hazard contribution returned by `hzr_phase_cumhaz()`.

Usage

```
hzr_phase_hazard(
  time,
  t_half = 1,
  nu = 1,
  m = 0,
  type = c("cdf", "hazard", "constant")
)
```

Arguments

`time`	Numeric vector of times (> 0).
`t_half`	Half-life parameter (> 0).
`nu`	Time exponent.
`m`	Shape parameter.
`type`	Phase type: `"cdf"` (early, uses $G(t)$), `"hazard"` (late, uses cumulative hazard from $h(t)$), or `"constant"` (flat rate, $\Phi = t$).

Details

- `"cdf"`: $\phi(t) = g(t)$ (density).
- `"hazard"`: $\phi(t) = h(t) = g(t)/(1 - G(t))$.
- `"constant"`: $\phi(t) = 1$.

Value

Numeric vector of instantaneous hazard contributions $\phi(t)$, same length as `time`.

See Also

`hzr_decompos()` for the underlying parametric family, `hzr_phase_cumhaz()` for the cumulative version.

Examples

```
t_grid <- seq(0.1, 10, by = 0.1)
phi_early <- hzr_phase_hazard(t_grid, t_half = 2, nu = 2, m = 0,
                              type = "cdf")
phi_late  <- hzr_phase_hazard(t_grid, t_half = 5, nu = 1, m = 0,
                              type = "hazard")
```

`hzr_read_outhaz` *Read a SAS* `outhaz` *estimate dataset*

Description

`PROC HAZARD`'s `outhaz=` dataset stores the converged estimates and the asymptotic variance-covariance matrix at full double precision, where the printed `.lst` carries about seven significant figures. For any quantity the dataset holds, it is the better parity reference: print precision stops being the binding constraint and optimizer convergence takes over.

Usage

```
hzr_read_outhaz(path)
```

Arguments

`path`	Path to a `.sas7bdat` written by `outhaz=`, or to an `.rds` holding the same data frame.

Details

The log-likelihood is *not* stored here; take it from the `.lst`.

Value

An `hzr_outhaz` object: a list with `estimates` (named numeric), `status` (named integer, 1 free / 0 fixed), `vcov` (matrix over free parameters, dimnames set) and `flags` (named numeric of model-structure flags). When no parameter is free, or the dataset was written with `NOCOV`, `vcov` is a 0x0 matrix rather than `NULL`, so check its dimensions rather than `is.null()`.

Experimental

This function is experimental and its return shape is expected to change. The result carries the fitted model, so `predict.hzr_outhaz()` predicts from it (that is what the `hzr_translate_sas(librefs = )` path emits), but it is not a `hazard` object and none of the other `hazard` methods apply to it. The `_STATUS_` coding is asserted against a synthetic fixture, so a real `OUTHAZ=` file using a different convention would yield an empty `vcov` alongside a fully populated `estimates`.

Examples

```
f <- system.file("extdata", "outhaz-fixture.rds", package = "TemporalHazard")
if (nzchar(f)) str(hzr_read_outhaz(f))
```

`hzr_repeated_events` *Build repeated-event segments, reproducing the SAS* `%repeat` *macro*

Description

Converts a long data set of candidate event times into one row per inter-event segment, ready to fit as a repeated-events model. This is a native R implementation of the SAS macro `%repeat` used by the repeated-events HAZARD jobs, whose fit input was seldom saved.

Usage

```
hzr_repeated_events(data, id, time, followup, indicator)
```

Arguments

`data`	A data frame with one row per candidate event per subject.
`id`	Name of the column identifying the subject.
`time`	Name of the column giving the interval from time zero to each event.
`followup`	Name of the column giving the interval from time zero to the end of follow-up.
`indicator`	Name of the event indicator column: 1 marks an event, 0 or `NA` marks its absence.

Details

The input holds one row per candidate event per subject, with an indicator naming the event of interest: a value of 1 marks an event, and 0 or `NA` marks its absence. For each subject the function keeps the event rows, guarantees at least one row, appends a censored row at the end of follow-up when the last event happened earlier, and then lags the event time within subject so that each row describes the interval since the previous event.

Two details are inherited from the macro and are not R conventions. An indicator code that is neither 0, 1 nor missing is treated as neither an event nor an absence, and rows are ordered with missing times first, as SAS sorts them.

The result can contain segments of zero length, where `iv_start` is not less than `time`: for example, an event that shares its time with the subject's previous row. They are kept, as the macro keeps them. A model that treats `iv_start` as the time a subject enters the risk set needs each segment to end after it starts, so adjust or remove these rows before such a fit. The SAS jobs this function reproduces moved `time` forward by about one hour on each.

`data` may not already contain a column named `rcensor`, `first`, `last`, `event`, `event_no`, `iv_start`, `iv_seg` or `renewal`: this function creates all eight and would silently overwrite an existing one of the same name.

Value

A data frame with one row per inter-event segment. Every input column is retained, with `time` and `indicator` altered where the macro alters them, and the following columns added:

`event` 1 when the row is an event of interest, otherwise 0.

`event_no` Running count of event repeats within the subject.

`rcensor` 1 when the row is right censored at the end of follow-up, otherwise 0. This is set both for a genuine censoring row and for an event row whose event time equals the end of follow-up. `rcensor` and `event` are therefore not mutually exclusive; see the Note below.

`iv_start` Interval from time zero to the start of the segment.

`iv_seg` Duration of the segment, `time` minus `iv_start`.

`renewal` Segment number under the modulated renewal formulation.

`first`, `last` 1 when the row was the subject's first or last row at stage 4, otherwise 0. The row stage 4 appends to pad a subject's censored tail inherits its source row's flags, so a subject can have two rows with `last == 1` together, or two with `first == 1` together.

Note

`rcensor` is not a plain censoring indicator: a row can have `event == 1` and `rcensor == 1` together, when a subject's last event happens to fall exactly at the end of follow-up. This is faithful to the SAS macro, whose own header warns that this combination is not compatible with a `HAZARD`-style fit. Do not pass `rcensor` directly as a censoring indicator without first accounting for the overlap with `event`.

Examples

```
# One row per candidate event per subject: s1 has two events, s2 one,
# and s3 none (its only row is a non-event, so it contributes no rows
# of its own before the function pads it with a censored one below).
events <- data.frame(
  id = c("s1", "s1", "s2", "s3"),
  t = c(1, 3, 2, NA),
  fu = c(10, 10, 10, 10),
  ev = c(1, 1, 1, 0)
)
# The result appends a censored row at end of follow-up for any subject
# whose last event happened before `fu`, so every subject gets at least
# one row even when, like s3, they never had an event.
hzr_repeated_events(events, id = "id", time = "t", followup = "fu", indicator = "ev")
```

`hzr_stepwise` *Stepwise covariate selection for a parametric hazard model*

Description

Run forward, backward, or two-way stepwise selection on an existing `hazard` fit using score (Q) statistics, Wald p-values, or AIC deltas as the entry / retention criterion. Phase-specific entry is supported for multiphase models: a covariate can enter one phase and not another.

Usage

```
hzr_stepwise(
  fit,
  scope = NULL,
  data,
  direction = c("both", "forward", "backward"),
  criterion = c("score", "wald", "aic"),
  slentry = 0.3,
  slstay = 0.2,
  max_steps = 50L,
  max_move = 4L,
  force_in = character(),
  force_out = character(),
  trace = TRUE,
  ...
)

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

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

## S3 method for class 'summary.hzr_stepwise'
print(x, ...)

## S3 method for class 'hzr_stepwise'
as.data.frame(x, ...)
```

Arguments

fit	A fitted hazard object built via the formula = Surv(...) ~ predictors, data = df interface, or a multiphase fit from either interface.
scope	Candidate set. NULL (default) uses every data-frame column not already in the model for every phase. A candidate enters as the column it names, whatever its name: a column literally named age:mal enters as that column, not the interaction, and a column "age " beside age enters as itself (#449). For single-distribution fits, pass a one-sided formula (~ age + nyha) or a character vector of names. Each name in a character scope is looked up, not parsed: a name that is exactly a column of data is that column, otherwise a name that is exactly a term label as terms() writes it ("`_X1`", "log(age)", "age:mal") is that term, and any other name is ignored with a warning naming it. The column is looked up first, so when data has a column literally named age:mal, "age:mal" puts that column in the scope rather than the interaction. A candidate enters as the column or term its name resolved to, so a bare "_X1" enters the column _X1 (#449, #441), and the score reads the values of that column however the name was written (#438). For multiphase fits, pass a named list of one-sided formulas keyed by phase, naming each phase once. scope lists what may enter; a drop consid-

ers every term in the model except `force_in` and terms frozen by `max_move` at an earlier iteration (see the **The frozen set** section). A two-sided formula is an error, since its left-hand side would never be a candidate, and so is a non-empty `scope` under `direction = "backward"`, which does not read it. An empty scope (`~ 1`, `character()`, or a list of `NULL`s and `~ 1`s) is accepted there.

`data` Data frame the base fit was built on. Required for refits. Its rows must be the fit's rows in the same order: the score criterion reads each candidate by position, so a sorted or reordered frame is refused when the fit stored the frame it was given. On the vector interface, and for any fit with `weights`, every criterion's refits pair `data` with vectors stored in the fit's row order, so there the frame is checked whatever the `criterion`. Columns added after the fit, such as derived candidates, are allowed. A vector-interface fit made without `data =`, or with a `data` of another row count used only to look names up, stores no frame to compare, and a `data` holding only columns added after the fit shares none; and shared columns with duplicate rows cannot show rows reordered among those duplicates. For those the order is checked against a column of `data` holding the fit's event times, and refused if they are out of order. Rows reordered within a tie leave that column unchanged, so when the times have ties the fit's other per-row inputs (status, interval bounds, weights and covariate design columns) are looked for in `data` too, each under the column it came from: the column the call named (`status = dead`, `status = d$dead`, or the event in `Surv(time, event)`), or a covariate's own name. Rows reordered within a tie are refused. The order is accepted when the inputs found tell every row apart, or when all of them are found: rows that could still be swapped are then identical in everything the fit reads, and the screen's answer is the same. With no column of event times, or tied times the other inputs cannot resolve, the order cannot be checked, and a warning of class `hzr_score_rows_unverified` says so and names the inputs to add. Column names must be unique: every check reads `data` by name, so a `data` with duplicated names is refused.

`direction` Search strategy: one of `"both"` (default), `"forward"`, or `"backward"`. Controls whether variables may only enter, only leave, or both. See the **Selection direction and criterion** section.

`criterion` Entry / retention rule: one of `"score"` (default), `"wald"`, or `"aic"`. `"score"` and `"wald"` both apply SAS-style p-value thresholds (`slentry` / `slstay`) but score entry candidates differently, and can therefore select different variable sets; `"score"` reproduces C/SAS HAZARD and needs no per-candidate refit, except at a step whose information cannot be formed, where it falls back to refitting and Wald-testing each candidate. `"aic"` adds or drops whenever it lowers the AIC. `"score"` and `"aic"` refuse a base fit that did not converge, since both measure candidates against it, and so does `"wald"` unless `direction = "forward"`, since a removal is tested on the base's own estimates. See the **Selection direction and criterion** section.

`slentry` Entry p-value threshold for the score / Wald criteria. Default `0.30` matches SAS `SLENTRY`.

`slstay` Retention p-value threshold for the score / Wald criteria. Default `0.20` matches SAS `SLSTAY`.

`max_steps` Limit on the rows of `$steps`: accepted actions and the `"frozen"` rows `max_move`

writes. It is checked before each iteration, so the iteration that reaches it can write several rows and pass it. Emits a `warning()` if hit. Default `50`.

`max_move` Per-variable oscillation cap. When a variable has entered + exited more than `max_move` times it is frozen for the remainder of the run, in the state it ends that iteration in. Default `4`. See the **The frozen set** section.

`force_in` Character vector of variables that must remain in the model. Such variables are still scored and reported in the selection trace, but are never dropped. A factor with several coefficients has no one score, so its row reads `NA`. Each name is looked up, not parsed, once, when the screen starts: a name that is exactly a column of `data` is that column, so the bare `"_X1"` pins the column `_X1` although `terms()` labels it `` `_X1` ``, and `"TRUE"` pins a column named `TRUE`. The exception is a column no model formula can hold, `"."` or `""`: it can never become a model term, so it cannot be pinned, and naming it warns and lists it in `unresolved`. Otherwise a name that is exactly a term label of the model or `scope` is that term, so `` "`_X1`" `` and `"age:mal"` work too. Any other name, `"age "` with a trailing space when there is no such column, say, matches nothing and is ignored with a warning naming it. The column is looked up first: when `data` has a column literally named `age:mal`, `"age:mal"` resolves to that column and not to the interaction.

`force_out` Character vector of variables that may never be considered as candidates. Names are looked up as for `force_in`: a column of `data` first, then a term label of the model or `scope`, and a warning for a name that is neither.

`trace` Logical; print step-by-step progress to the console. Default `TRUE`.

`...` Passed to every candidate refit. Only `control` (e.g. `control = list(maxit = 500)`) and an `objective` equal to the base fit's are accepted. Any other name is an error: a misspelling such as `slentyr` would be stored by `hazard()` without being read, and every other `hazard()` argument (the response, `weights`, `time_windows`, `dist`, `theta`, `phases`, `fit` and so on) is set by the refit itself, from the base model and `data`, so that each candidate is compared with the model it extends. Every candidate refit, and the selected model, runs under the elements of the base fit's `control` that a fit reads (`maxit`, `reltol` and, for a multiphase fit, `n_starts`, `conserve`, `phase_share_tol` and `start_seed`), as every step of a SAS `SELECTION` runs under one `PROC HAZARD`'s options. A `control` given here overrides it entry by entry: `control = list(maxit = 500)` changes `maxit` and keeps the rest. The `print()`, `summary()` and `as.data.frame()` methods ignore `....`

`x` An `hzr_stepwise` object.

`object` An `hzr_stepwise` object.

Details

The `steps` data frame has columns:

`step_num` Integer sequence starting at 1.

`action` `"enter"`, `"drop"`, or `"frozen"`.

`variable` The term entered, dropped or frozen, as the model's `terms()` label: a non-syntactic column `_X1` is recorded as `` `_X1` `` on every row, whether the scope named it `"_X1"`, `"`_X1`"` or in a formula.

`phase` Phase name (multiphase) or `NA_character_`.

`criterion` The criterion actually applied to this step: `"score"`, `"wald"`, or `"aic"`. Under `criterion = "score"` the drop rows read `"wald"`, because score is entry-only.

`score` Winning score used for the decision.

`stat`, `df` Test statistic and degrees of freedom.

`stat_type` What `stat` is on this row, and so which reference distribution recomputes its p-value: `"score_q"` (chi-square on `df`), `"wald_z"` (standard normal) or `"wald_chisq"` (chi-square on `df`). `df` alone does not distinguish them; a scalar Wald is reported as a z, not as its square, so it and a score Q are both recorded at `df = 1` while calling for different distributions. It also identifies the rows the Wald fallback rescued, but only among *entry* rows: under `criterion = "score"` those are the rows with `action == "enter" & stat_type == "wald_z"`. Drop rows are always Wald-tested under that criterion (removal follows SAS and is tested on the current model's Wald p-value), so they read `"wald_z"` whether or not the fallback ever fired. See `$criteria$n_wald_fallbacks` for the count.

`p_value`, `delta_aic` Always populated when computable, regardless of the active criterion.

`logLik`, `aic`, `n_coef` Goodness-of-fit diagnostics of the model *after* this step. Under `objective = "sas"`, on a step whose model reads an interval-censored row (one of positive weight, not dropped by a phase design), `logLik` holds PROC HAZARD's objective, not a log-likelihood, and `aic` is computed from it. A step can change which rows are read (see `n_rows`), so on a step that starts or stops reading such rows `delta_logLik` is the difference of two different quantities. The trace's final line labels the final model's value, and `criterion = "aic"` warns once (class `hzr_stepwise_sas_objective`) as soon as a step's model reads such a row.

`n_rows` Number of rows in the fit's data after this step (rows given weight 0 are counted). A multiphase fit drops every row where a variable in the model is missing, so entering a variable with missing values shrinks the sample every later step is tested on, and dropping it grows the sample back. A step that changes the rows of positive weight raises a warning of class `hzr_stepwise_sample_changed` naming the step, the variable and the row counts before and after. Which variables enter is not changed by this: compare `n_rows` across steps to see how many rows each test rested on.

Value

An object of class `c("hzr_stepwise", "hazard")`, the final fit augmented with:

`steps` Data frame with one row per accepted / frozen action; see Details.

`scope` Record of the candidate scope, plus `force_in`, `force_out`, and the frozen set. Four fields answer four different questions about the pins, and none substitutes for another: `force_in` and `force_out` are the names **as given**, including any that pinned nothing; `force_in_resolved` and `force_out_resolved` are what those names **resolved to**, as a column or term label, so a bare `"_X1"` appears there as `` "`_X1`" ``; `unresolved` is which of them resolved to nothing; and `frozen` is what the loop held in that the caller never named. The resolved fields carry the resolved names only: the frozen set is **not** merged into them, because `frozen` already records it and merging would list variables nobody asked for (#451). Resolving is not applying: a

name that resolves to a variable the model does not contain is recorded here and still pins nothing, because `force_in` only keeps a variable that is already in. The resolved fields are **not** aligned element for element with the as-given ones, which stay longer by every name that resolved to nothing. A variable in `frozen` is in the final model if it was in the model when it was frozen, and out of it otherwise; see the **The frozen set** section. `unresolved` is a list with elements `force_in`, `force_out` and `scope`, each the names that could not be used and were therefore ignored (`character()` when none were). That is usually a name matching neither a column of `data` nor a term label; for `force_in` and `force_out` it also covers a name that IS a column but which no model formula can hold, such as `"."` or `""`, since such a column can never become a model term and so can never be pinned. The trace, and so `print()` and `summary()`, carries a line for each non-empty one, and a screen whose character `scope` was emptied this way says so where it stops. `candidates`, `force_in` and `force_out` here are the arguments as given, so a character `candidates` still lists the names that were ignored, as `force_in` and `force_out` do; read `unresolved` for which those were (#451).

`criteria` Named list of the threshold / direction settings actually applied, plus `n_uncomputable_scores` (how many candidate scores were `NA`, counted once per step: an entry the score test could not score under `criterion = "score"`; an entry under `criterion = "wald"`, or a removal under any criterion, whose Wald statistic could not be computed for want of a variance, `wald_no_variance`; or an entry under `criterion = "aic"` whose fit had no finite objective, `nonfinite`, or was fitted on different rows from the current model because the candidate is missing on some, `rows_differ`; or an entry under `criterion = "aic"` or `"wald"`, or rescued by the Wald fallback under `"score"`, whose refit ended below the current model's log-likelihood, which it contains, so that the refit cannot have converged, `loglik_below_base`), `uncomputable_reasons` (a named integer vector of *why*), `wald_untested_removals` and `wald_untested_entries` (the `"var"` / `"var@phase"` tokens of variables kept in, or left out, on a step whose Wald test for them could not be computed; a variable tested, or refused as `loglik_below_base`, at a later step is not listed. Entries are listed under `criterion = "wald"` only: under `"score"` an entry no test could reach is reported by its reason, such as `fallback_no_variance`) and `stopped_uncomputable` (`TRUE` when the last iteration had candidates for entry or for removal and could test none of them). Read `uncomputable_reasons` before treating an unscored candidate as a bad one: `information_indefinite` marks candidates whose effect is too large for the score test's approximation at zero, which are typically the strongest variables on offer rather than degenerate ones. Candidates with that cause, or with `coefficient_diverging`, `information_nonpositive` or `nuisance_singular`, are refit and tested by Wald automatically, counted in `n_wald_fallbacks`. `nuisance_singular` is a fault of the score test at the current model, whose information matrix could not be formed or inverted, so it applies to every candidate at that step and each of them is refitted. A candidate still reaches `uncomputable_reasons` when that refit fails, or when its cause is any other, for which no refit is attempted. Read `uncomputable_reasons` for which one it was in any given run. For every criterion it also carries `refit_failures` (the `"var"` / `"var@phase"` tokens of candidate moves whose refit errored, failed to converge, or was refused because the move would not change the model – a drop that removes no design column, #320), `refit_failure_reasons` (why each one failed or was refused: the refit's error message, that it did not converge, or that the move changes nothing; named by the same tokens), `n_refit_failures`, and `stopped_refit_failed` (`TRUE` when the run ended on an iteration in which a refit failed or a move was refused. A refit failure is a screen that could not test its candidates, rather than one that tested them and liked none; a refusal is determinate – the move was tested and would have left the model as it was. Read `refit_failure_reasons` for

which it was). Check it before reading a zero-row `steps` as an honest null result.

`trace_msg` Character vector of the trace lines, captured regardless of the `trace` flag.

`elapsed` `difftime` from start to finish.

`final_call` The call that produced this result.

`print.hzr_stepwise` returns `x` invisibly.

`summary.hzr_stepwise` returns a `summary.hzr_stepwise` object (extends `summary.hazard`) with `$stepwise_steps` and `$stepwise_trace` appended.

`print.summary.hzr_stepwise` returns `x` invisibly.

`as.data.frame.hzr_stepwise` returns the `$steps` data frame.

Selection direction and criterion

Two arguments shape the search. `direction` decides which moves are allowed at each step; `criterion` decides how a candidate move is scored and whether it is accepted.

`direction = "forward"` Start from the base model and only *add* variables; the best eligible candidate enters each step until none clears the entry rule. Variables never leave once in.

`direction = "backward"` Start from the base model, which must already hold every candidate, and only *drop* variables; the weakest term leaves each step until all survivors clear the retention rule. `scope` is not read, so a non-empty one is an error: protect terms with `force_in`.

`direction = "both"` **(default)** Two-way stepwise: on every iteration, whether or not a variable entered, every term in the model, the base model's included, is re-tested and may be dropped unless it is in `force_in` or was frozen by `max_move` at an earlier iteration (see the **The frozen set** section). `scope` limits what may enter, not what may leave. This is the SAS `SELECTION = STEPWISE` strategy. `max_move` caps how often a single variable may oscillate before it is frozen.

`criterion = "score"` **(default)** Accept moves on SAS-style significance thresholds, using the score (Q) statistic of the candidate coefficient; this reproduces C/SAS HAZARD's `SELECTION` statistic. Q is evaluated at the *current* model's MLE with the candidate's coefficient pinned at zero, so **no candidate refit is needed**: the reduced-model information is inverted once per step and reused across every candidate. Only the winner is refit. A candidate enters if its p-value is below `slentry`.

Score is an *entry* criterion; the drop path never refit per candidate in the first place, so removals are tested on the current model's Wald p-value against `slstay`, as SAS does.

For single-distribution fits, the score criterion computes the observed information numerically via the suggested **numDeriv** package and errors with a clear message if it is not installed; a multiphase fit uses the analytic Hessian instead and does not need it.

Following SAS, the variance used during *selection* is approximate: shaping-parameter covariances are ignored. This affects selection only; final-model standard errors are unchanged and still come from the full Hessian. Candidates must be single-column numeric main-effect terms; a factor is rejected with an error rather than skipped.

`criterion = "wald"` Accept moves on SAS-style significance thresholds, using the Wald χ^2 of the affected coefficient(s): a candidate enters if its p-value is below `slentry`, and a term is dropped if its p-value rises above `slstay`. Entry candidates are scored from a refit that adds

the candidate (so its new coefficient can be tested); drop candidates are scored from the *current* model's Wald p-values without a per-candidate refit, and a single refit is run only after a drop is chosen. This was the default before version 1.2.0. It differs algorithmically from C/SAS HAZARD, so the two criteria can take different step paths (and select different variable sets) even when they converge to a similar final model.

`criterion = "aic"` Accept any move with $\Delta\mathrm{AIC} < 0$ (a strictly better penalised fit), ignoring `slentry` / `slstay`. Entry candidates use the actual $\Delta\mathrm{AIC}$ from the candidate refit; drop candidates use a Wald-to-likelihood-ratio approximation, $\Delta\mathrm{AIC} \approx W - 2\,\mathrm{df}$, computed from the current model without a per-candidate refit (the chosen drop is refit afterwards). Use this for a non-significance-based, information-criterion search.

The frozen set

A variable that has moved (entered or left) more than `max_move` times is frozen: it is held in the state it is in, in the model or out of it, for the rest of the run, and listed in `$scope$frozen`.

The freeze takes effect at the end of the iteration in which the variable reached the cap. A two-way iteration (`direction = "both"`) makes a forward step and then a backward step, so a variable that reaches the cap by entering and is dropped by the same iteration's backward step is frozen **out**. Its `"frozen"` row in `$steps` follows the `"drop"`. A variable is therefore in the final model exactly when it was in the model when it was frozen, and `$scope$frozen` agrees with the final model.

Earlier versions froze such a variable as it entered and then dropped it anyway, so `$scope$frozen` could name a variable the final model did not contain (#580).

This is not PROC HAZARD's MOVE rule. PROC HAZARD makes one move per step, a removal before any entry; it counts a variable's exits (and, under `NOSTEPWISE`, its entries too), separately for each phase; and a variable at its limit can neither leave nor enter. Outside `NOSTEPWISE` it therefore freezes a variable only as it leaves. `max_move` counts entries and exits alike, by name across phases, which is why the SAS translator records `MOVE=` rather than mapping it. Adopting PROC HAZARD's rule would change which variables are selected, and is deferred; see https://github.com/ehrlinger/TemporalHazard/issues/378 and https://github.com/ehrlinger/TemporalHazard/issues/379.

See Also

`hazard()` for the base model and `hzr_phase()` for multiphase scopes; `stepwise_trace()` to retrieve the captured selection log.

Examples

```
data(avc)
avc <- na.omit(avc)
base <- hazard(survival::Surv(int_dead, dead) ~ age,
               data = avc, dist = "weibull", fit = TRUE,
               theta = c(mu = 0.01, nu = 0.5, 0))

sw <- hzr_stepwise(base, scope = ~ age + mal,
                   data = avc, direction = "forward")
print(sw)
```

`hzr_theta_names` *Parameter names for a phase specification, in* `theta` *order*

Description

Returns the names of the `theta` vector a multiphase `hazard()` fit builds from `phases`, in the order `theta` requires. Use it to check a hand-written starting vector against the specification it is meant to go with, before any fit runs.

Usage

```
hzr_theta_names(phases, covariates = NULL)
```

Arguments

`phases`	A non-empty list of `hzr_phase()` objects, as passed to `hazard()`.
`covariates`	Optional named list mapping phase name to that phase's covariate column names, e.g. `list(early = c("age", "sex"))`. Phases absent from the list are treated as having no covariates. Names are used verbatim, so they must match the columns the fit will see.

Details

`theta` is positional and its entries are not on a common scale. For an `early` + `late` pair the layout is

```
early.log_mu, early.log_t_half, early.nu, early.m,
late.log_mu,  late.log_tau,     late.gamma, late.alpha, late.eta
```

so the late phase logs `mu` and `tau` while carrying `gamma`, `alpha` and `eta` on the natural scale. **Wrapping the wrong element in `log()` produces a fit, not an error**, which is why a comment describing the order is not enough and this returns the real thing. The order is a property of the specification, so it changes the moment a phase is added, removed or retyped.

Phase names come from `names(phases)`; unnamed phases are labelled `phase_1`, `phase_2`, ... by the same validation `hazard()` applies, so the labels here are the labels a fit will use.

Phase names must be unique, and `"total"` and `"time"` are reserved; this applies the same validation `hazard()` does, so both reject them identically.

Value

A character vector whose **order is the required `theta` order**. Its length is the number of parameters the specification implies, so `length(hzr_theta_names(phases))` is the length `theta` must have.

See Also

`hzr_phase()` for building a specification, `hazard()` for the fit whose `theta` this names.

Examples

```
phases <- list(early = hzr_phase("cdf"), late = hzr_phase("g3"))
hzr_theta_names(phases)

# Check a hand-written starting vector before fitting.
theta0 <- c(log(0.05), log(0.2), 0, -0.4, log(0.03), log(1), 1, 1, 1)
stopifnot(length(theta0) == length(hzr_theta_names(phases)))
setNames(theta0, hzr_theta_names(phases))

# With covariates on one phase.
hzr_theta_names(phases, covariates = list(early = c("age", "sex")))
```

---

hzr_translate_sas *Translate a SAS HAZARD job into a Quarto document*

Description

Reads a SAS program containing PROC HAZARD and/or PROC HAZPRED blocks and emits a Quarto document of the equivalent hazard() and predict.hazard() calls.

Usage

```
hzr_translate_sas(path, out_dir = NULL, librefs = NULL)
```

Arguments

path	Path to a .sas file.
out_dir	Directory to write the .qmd into. NULL (default) parses without writing.
librefs	Optional named character vector mapping SAS librefs to directories, e.g. c(EX = "estimates"), used to resolve INHAZ=. The resolved member is read as <member>.sas7bdat; give a librefs value that already ends in .rds or .sas7bdat (a specific file, not a directory) to name an already-converted fit directly, e.g. c(EX = "estimates/hzdeath.rds").

Details

Experimental: a job that translates does render (the emitted hazard() chunk binds its fit and asks for an actual fit), but this is a translation aid, not a turnkey reproduction, and some SAS constructs are refused rather than translated. See the Experimental section below.

Constructs the translator does not cover are recorded on the returned object and rendered as visible callouts, never dropped. A PROC HAZPRED job whose INHAZ= fitted model cannot be located emits a stop(), so the document fails to render rather than reporting over a model it did not load.

A phase-statement variable that PROC HAZARD cannot read as a name, such as AGE*SEX, LOG(AGE) or B SEX, is left out of the model. (LOG() is read as LOG, as PROC HAZARD reads it: its lexer treats) as whitespace and (as a change of state, hazard_1.1:32 and :56.) A phase variable

must be a NAME, `[_A-Z][_A-Z0-9]*` (`hazard_l.l:39`; `phasevar : NAME`, `hazard_y.y:213`), so `PROC HAZARD` rejects such a job at parse. The document raises a `warning()` above the fit, and the operand is recorded in `$untranslated`.

Two more syntax errors take the same route. A value on a `PROC HAZARD` option that takes none, such as `NOCOV=1` (`hazard_y.y:65-76`), is ignored in the fit. A `TIME`, `EVENT`, `RCENSOR`, `LCENSOR` or `WEIGHT` statement with more than one operand, which takes exactly one name (`hazard_y.y:106-127`), is fitted on the first. Both are warned about above the fit and recorded in `$untranslated`. `TIME` or `EVENT` with no operand, and no other statement supplying the variable, leaves nothing to fit, so that job's fit chunk is a `stop()`.

A job may contain more than one `PROC HAZARD` and/or `PROC HAZPRED` block. Every block is preserved: the first of a kind keeps the bare chunk name (`fit`, `pred`, `pred_haz`), later ones get `fit_2`, `fit_3`, `pred_2`, and so on, so the emitted document contains every model, not just the last one seen. `job$outhaz` is therefore a character vector, one element per `PROC HAZARD` block that set `OUTHAZ=` (in the order the blocks appear). Each `PROC HAZPRED` block's `INHAZ=` is resolved independently against that vector, matching the most recently written `OUTHAZ=` at that point in the file, mirroring SAS itself, where a later `OUTHAZ=` write overwrites the dataset an earlier one wrote under the same name. If a job's own `OUTHAZ=` values don't cover it, `librefs` is tried next; distinct external `INHAZ=` values each get their own loaded-fit chunk. When neither resolves and the job holds more than one local fit, which fit a `predict()` call belongs to is genuinely unknown; the emitted call falls back to referencing `fit` and the ambiguity is recorded in `untranslated`, never guessed at silently.

Value

An `hzr_sas_job` object, invisibly. `$calls` holds the emitted calls keyed by chunk label, `$grid` the last prediction grid seen and `$inhaz` the first unresolved `INHAZ=` (not all of each, when a job has several), `$untranslated` the recorded gaps, `$coverage` the token counts, `$notes` any callout attached to a chunk by label, emitted immediately above that chunk in the rendered document, and `$not_run` the chunks SAS never ran, named by label, each holding the `%MACRO` the job defines but never calls around it. Those chunks are rendered with `eval: false`; a caller evaluating `$calls` itself should skip them.

What the emitted document needs to run

Base R at the version this package declares in `DESCRIPTION` (`Depends`), plus TemporalHazard itself. The emitted chunks never reach into this package's internals, so the document renders in your session, and in a colleague's, without anything further installed, with one exception. A translated `SELECTION` screen on a multiphase job with interval-censored rows (an `ICENSOR` job; `LCENSOR` is left truncation and does not need it) needs the suggested package **numDeriv**. Without it a screen that tests an entry stops and says so. A screen that completes but cannot test a removal (a `BACKWARD` screen, whose base has no usable variance) is caught by the emitted check, which warns and names the variables.

Experimental

The emitted document renders: the `hazard()` chunk binds its fit to a name and passes `fit = TRUE`, so the `predict()` chunks have something to predict from. A `SELECTION` statement is translated into an `hzr_stepwise()` call carrying the job's own candidates, per-variable flags and thresholds

(#160); the screen is real, and may select a different model than PROC HAZARD did, which the emitted document says in a callout above the chunk. SELECTION options with no faithful translation are refused outright rather than mistranslated, emitting a stop() in place of the fit: FAST, MAXVARS, RESTRICT, a per-variable MOVE= or ORDER=, and a variable held by /I in one phase but movable in another. So is LCENSOR combined with ICENSOR, which one time_lower argument cannot express (#155). A job with ICENSOR is fitted with objective = "sas", the interval term PROC HAZARD accumulates, so it reproduces PROC HAZARD's estimates; the value it reports is then PROC HAZARD's objective, not a log-likelihood, and a callout above the chunk says so (#543). Degenerate intervals are resolved as PROC HAZARD does before its fit: a lower bound equal to the time makes the row an exact event, and one that is missing, negative or after the time drops the row from the fit (not from your data frame); the status chunk warns with both counts. Prediction grids the parser cannot resolve are refused whole, and the predict() chunks that would have read such a grid become a stop() naming it, rather than a predict(newdata =) over a name no chunk builds. An unresolved INHAZ= stops the render on purpose.

On a fit loaded from an external INHAZ= dataset, point predictions work but se.fit = TRUE is refused when PROC HAZARD estimated a late shape parameter on a composite scale (the generic unconstrained three-phase case, not an exotic one). A translated PROC HAZPRED block asks for confidence limits unless the SAS job says NOCL, so such a job stops at its predict() chunks.

Repeated events

A job that builds its fit input with the SAS macro %repeat has that call translated to hzr_repeated_events(). The emitted chunk renames the function's output columns to the names the job gives them, uppercased like every name the translator emits, so the fit reads them. An input column named like one of those outputs is dropped first, with a warning, because the macro overwrites it. The macro's input is built by the job's own DATA steps, which are not translated, so the document stops until that input is assigned. Any step between the macro and the fit that names the macro's output, or uses a macro variable that might, is not translated either; a plain PROC SORT that only reorders rows is the one step let through. Its chunk stops and quotes the step, for the reader to replace with R code or to delete if the step leaves the output unchanged. The same holds between two %repeat calls when the second reads the first's output. A step that changes the output without naming it, such as a macro that writes it internally, is not detected.

Comparing a translated fit against a SAS listing

The emitted hzr_phase("g3", ...) call carries the TAU, GAMMA, ALPHA and ETA that PARMS specified. PROC HAZARD does not always report the values it was given: when ALPHA is fixed at 1 it rewrites the late phase before fitting, pinning TAU at 1 and folding the two shape parameters into one exponent, because at alpha = 1, tau = 1 the G3 form collapses to $t^{\gamma\eta}$ and only that product is identified. With ETA fixed it reports GAMMA as $\gamma\eta$ and ETA as 1; otherwise it reports ETA as $\gamma\eta$ and GAMMA as 1.

So a listing can print a GAMMA/ETA pair that differs from the job's own PARMS text, and from the emitted call, while describing the same fitted model. Compare the **product** $\gamma\eta$, not the two parameters separately, and expect TAU to read 1 and fixed. The log-likelihood, MUL and every prediction agree regardless. Most jobs are unaffected in practice: GAMMA = 1 FIXGAMMA is the usual companion to ALPHA = 1 FIXALPHA, and it makes the rewrite an identity.

The one case that is a genuine model difference (ALPHA fixed at 1 with GAMMA and ETA *both* estimated, where PROC HAZARD fixes ETA and fits one parameter fewer than hazard() would) is

recorded in `$untranslated` rather than left to be discovered in the comparison.
`$coverage` counts tokens the parser recognised; it is not evidence that the emitted calls execute, and a job can report full coverage with an empty `$untranslated` while its document still errors on render. See the 1.2.2 `NEWS.md` entry. The function's API, the `hzr_sas_job` field layout and the emitted document format are all expected to change.

Examples

```
job <- hzr_translate_sas(
  system.file("extdata", "hz-example.sas", package = "TemporalHazard")
)
```

`is_hzr_phase` *Test if an object is an hzr_phase*

Description

Test if an object is an hzr_phase

Usage

```
is_hzr_phase(x)
```

Arguments

`x`	Object to test.

Value

Logical scalar.

Examples

```
is_hzr_phase(hzr_phase("cdf"))
is_hzr_phase("not a phase")
```

omc *OMC: Open Mitral Commissurotomy*

Description

Data for 339 patients who underwent open mitral commissurotomy at the University of Alabama Birmingham. Records repeated thromboembolic events (up to 3 per patient) as indicators; the event dates are not included.

Usage

```
omc
```

Format

A data frame with 339 rows and 7 variables:

study Patient identifier

te1 Indicator for first thromboembolic event

te2 Indicator for second thromboembolic event

te3 Indicator for third thromboembolic event

int_dead Follow-up interval to death or last contact (months)

dead Death indicator (1 = dead, 0 = censored)

opdjul Operation date (Julian)

Source

University of Alabama Birmingham cardiac surgery registry.

See Also

Other datasets: `avc`, `cabgkul`, `tga`, `uslife2023`, `valves`

predict.hazard *Predict from a hazard model object*

Description

Produces prediction outputs from a `hazard` object. Supports multiple prediction types including linear predictor, hazard, survival probability, and cumulative hazard.

Usage

```
## S3 method for class 'hazard'
predict(
  object,
  newdata = NULL,
  type = c("hazard", "linear_predictor", "survival", "cumulative_hazard"),
  decompose = FALSE,
  se.fit = FALSE,
  level = 0.95,
  conf.type = c("log-log", "logit"),
  ...
)
```

Arguments

object	A hazard object. One built with fit = FALSE holds the starting values it was given rather than estimates, so predicting from it warns (condition class hzr_unfitted_prediction); under dist = "multiphase" it is an error instead, because the per-phase designs are resolved only when the model is fitted. To evaluate a model at parameters you supply, use hzr_evaluate(). The warning is governed by options(TemporalHazard.warn_unfitt =), which this package's own tests set to FALSE where they exercise that capability deliberately; leaving it on is what tells a reader that a number came from a starting value.
newdata	Optional matrix or data frame of predictors. For types requiring time (e.g., "survival", "cumulative_hazard"), newdata should include a time column, or time will be taken from the fitted object's data. Covariates are matched to the model by column name, so their order does not matter. A formula fit rebuilds its designs from the formulas, global and per phase, with the factor levels and contrasts the fit saw, so a factor can be given as a level label. newdata may instead carry the fit's design-matrix columns by name (grpyoung for a factor grp); these are used only when no formula variable is given (a numeric variable that is itself a column counts only if another column, such as I(age^2), is built from it). With all the variables given, the design is rebuilt from them, so a design column that contradicts one is ignored; some variables beside the design columns, with others missing, is an error. That error is conservative in two cases where nothing contradicts: poly() design columns given with the variable they are built from but without another variable, and a non-syntactic name such as my age given both as the design column `my age` and as the variable. Give all of the formula's variables, or only the design columns. A column the model does not use is ignored, and a covariate the model needs but newdata lacks is an error. Only the columns of the model's data are taken from newdata: a formula constant (cutoff in I(age > cutoff), spline knots) comes from the formula's environment, and a term that uses row-level values kept outside data (~ zz, with zz a vector in the workspace) is an error, even when newdata has a zz column; move it into data and refit. A term that computes a statistic over newdata's rows warns only for the functions the check knows (see "How newdata is evaluated"). Any other function, including one you write, is still recomputed from newdata's rows, silently: no warning means undetected, not safe. A fit made with an unnamed x

matrix matches by position. For the types requiring time, a newdata with only a time column evaluates the baseline, with every model-matrix column at 0. For age that is an age of 0; for scale(age) it is the mean age. predict() warns, once per call and with class "hzr_predict_covariates_zero", naming the data variables to supply (age, not scale(age)). Supply them in newdata to predict at other values, or muffle that class when the baseline is what you want. A model without covariates does not warn.

For those types, the time column of newdata is always the prediction time, so a model whose formula uses a variable named time (a covariate, or a constant such as I(age > time)) cannot be given those types at newdata; rename it and refit.

type — Prediction type:

- "linear_predictor": Linear predictor eta = x*beta (not available for multiphase)
- "hazard": Instantaneous hazard. Single-distribution models return the hazard scale exp(eta); multiphase models return the additive hazard h(t|x) = sum_j mu_j(x) phi_j'(t) and so require time values (like "survival"/"cumulative_hazard"). decompose is not supported for "hazard".
- "survival": Survival probability S(t|x) = exp(-H(t|x))
- "cumulative_hazard": Cumulative hazard H(t|x) at event times

decompose — Logical; if TRUE and the model is multiphase, return a data frame with per-phase cumulative hazard contributions alongside the total. Ignored for single-distribution models. Default FALSE.

se.fit — Logical; if TRUE, compute delta-method standard errors and confidence limits for each prediction. The return value becomes a data frame with columns fit, se.fit, lower, upper. Default FALSE. CLs are computed on the log-hazard / log-cumhaz scale and on the log(-log(survival)) scale so lower/upper stay inside the valid range of each prediction type; linear_predictor uses symmetric natural-scale CLs. For multiphase models, se.fit = TRUE combines with decompose = TRUE when type = "cumulative_hazard": the result is a long data frame with one row per prediction time and component (component in "total" plus each phase name) and columns fit, se.fit, lower, upper. Per-phase CLs use only that phase's parameters, so they do not sum to the total CL. The combination is not available for type = "survival" (per-phase survival is not additive).

level — Numeric confidence level in (0, 1); default 0.95. Only used when se.fit = TRUE.

SAS draws narrower bands than this by default. PROC HAZPRED takes its width from CLIMITS=, whose default is 0, and any value outside (0, 1) gives a multiplier of exactly one, so the band is one standard error, 68.3%, not 95%. Reproducing a SAS figure at this function's default therefore yields a band about 1.96 times wider than the one being checked against, with no error and no warning on either side. Pass the SAS level explicitly to match:

```
predict(fit, newdata, type = "survival", se.fit = TRUE,
        level = 2 * stats::pnorm(1) - 1, conf.type = "logit")
```

The default is left at `0.95` deliberately: it is the right R-side default, and silently adopting SAS's would make this method disagree with every other R modelling function.

`conf.type` Transform for `type = "survival"` confidence limits when `se.fit = TRUE`: `"log-log"` (default) builds them on `log(-log S)` (the `survival::survfit` standard); `"logit"` builds them on `logit(1 - S)`, reproducing SAS HAZARD's `HAZPRED` survival limits. Other types are unaffected (hazard/cumulative-hazard use a log scale that already matches HAZPRED). Only used when `se.fit = TRUE`.

`...` Unused; included for S3 compatibility.

Details

For Weibull models with survival or cumulative_hazard predictions:

- Cumulative hazard: H(t|x) = (mu*t)^nu * exp(eta)
- Survival: S(t|x) = exp(-H(t|x))

Time values must be positive and finite. If newdata contains a `time` column, it will be used; otherwise, the time vector from the fitted object is used. For models fit with `time_windows`, predictions for `type = "linear_predictor"` or `"hazard"` also require time values (via `newdata$time` or fitted-time fallback) so window-specific coefficients can be selected.

See the section "How `newdata` is evaluated" for what is recomputed from `newdata` and when `predict()` warns.

Value

When `se.fit = FALSE` (default), a numeric vector of predictions. When `se.fit = TRUE`, a data frame with columns `fit`, `se.fit`, `lower`, `upper` (delta-method point estimate, standard error, and confidence limits at `level`). `se.fit` is always the standard error of `fit` on its own scale: for `type = "survival"` that is `S * se(H)`, the delta method applied to `S = exp(-H)`. The survival limits are built from `se(H)` on the `conf.type` scale, so they are not `fit +/- z * se.fit`. `PROC HAZPRED` prints no standard error, only the limits. For multiphase `type = "cumulative_hazard"` with `decompose = TRUE`, a long data frame (`time`, `component`, `fit`, `se.fit`, `lower`, `upper`); with `decompose = TRUE` and `se.fit = FALSE`, a wide data frame of per-phase contributions.

How `newdata` is evaluated

`predict()` evaluates the model's formulas on `newdata` as given, as `stats::predict.lm()` does. It uses the fit's factor levels and contrasts, and the centering, basis and knots that a top-level `scale()`, `poly()`, `ns()` or `bs()` term recorded. Everything else is recomputed from `newdata`, so a prediction can differ from the fit without any error. `predict()` warns, naming the cause, in three such cases. The predicted values are the same with or without the warning.

- **A term that computes a statistic over the rows.** In `I(age - mean(age))`, `I(scale(age)^2)` or `I(as.integer(factor(grp)))`, the mean, the scaling or the factor coding comes from `newdata`'s rows, so a row's prediction depends on which other rows are given. Compute such a variable in the data before fitting, and supply it in `newdata`. The check knows a fixed list of R's functions, among them `mean()`, `median()`, `min()`, `max()`, `quantile()`, `sd()`, `IQR()`, `ave()`, `rank()`, `length()`, `scale()`, `factor()` and `cut()` with a count of breaks. A function

not on it, including one you write, is recomputed from newdata's rows just the same, with no warning: the list is a floor, not a boundary.

- **A column of another type than the fit saw.** A numeric column given as character compares as text ("154.6" > 50 is FALSE), and a difftime in other units is used in those units. The check compares against the fitting data the fit kept, which fits saved before TemporalHazard 1.1.0 do not have, so those fits are not checked. A factor given as its level labels, or an integer for a double, is not a mismatch.
- **A design rebuilt under this session's contrasts.** A formula fit saved by version 1.2.10 or earlier kept no record of its contrasts, and its design is rebuilt under options(contrasts =). predict() warns when that option names a function other than contr.treatment or contr.poly and the rebuilt design is used. A redefined contr.treatment is not detected.

Two cases are not detected:

- A constant the formula reads from its environment, such as cutoff in I(age > cutoff), is read when you predict, so a value changed since the fit is used.
- A comparison of strings, such as I(grp > "b"), follows the session's collation (LC_COLLATE), which can order strings differently from the session that fitted the model.

See Also

hazard() for model fitting, summary.hazard() for model summaries, hzr_phase() for multi-phase temporal shapes.

vignette("prediction-visualization") for detailed prediction workflows including decomposed hazard plots and patient-specific curves.

Examples

```
# -- Basic predictions -------------------------------------------------
set.seed(1)
fit <- hazard(time = rexp(50, 0.3), status = rep(1L, 50),
              theta = c(0.3, 1.0), dist = "weibull", fit = TRUE)
predict(fit, type = "survival")
predict(fit, newdata = data.frame(time = c(1, 2, 5)),
        type = "cumulative_hazard")

# -- Patient-specific survival curves ----------------------------------
set.seed(1001)
n   <- 180
dat <- data.frame(
  time   = rexp(n, rate = 0.35) + 0.05,
  status = rbinom(n, size = 1, prob = 0.6),
  age    = rnorm(n, mean = 62, sd = 11),
  nyha   = sample(1:4, n, replace = TRUE),
  shock  = rbinom(n, size = 1, prob = 0.18)
)
fit2 <- hazard(
  survival::Surv(time, status) ~ age + nyha + shock,
  data  = dat,
  theta = c(mu = 0.25, nu = 1.10, beta1 = 0, beta2 = 0, beta3 = 0),
```

```
  dist  = "weibull", fit = TRUE
)

new_patients <- data.frame(
  time = c(0.5, 1.5, 3.0),
  age  = c(50, 65, 75),
  nyha = c(1, 3, 4),
  shock = c(0, 0, 1)
)
# Compute predictions from the clean covariate frame before adding columns
surv   <- predict(fit2, newdata = new_patients, type = "survival")
cumhaz <- predict(fit2, newdata = new_patients, type = "cumulative_hazard")
new_patients$survival          <- surv
new_patients$cumulative_hazard <- cumhaz
new_patients


# -- Grouped survival curves ----------------------------------------
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)

  t_grid <- seq(0.05, max(dat$time), length.out = 80)
  profiles <- data.frame(
    label = c("Low risk (age 50, NYHA I)",
              "High risk (age 75, NYHA IV)"),
    age   = c(50, 75),
    nyha  = c(1, 4),
    shock = c(0, 1)
  )

  curve_list <- lapply(seq_len(nrow(profiles)), function(i) {
    nd <- data.frame(
      time  = t_grid,
      age   = profiles$age[i],
      nyha  = profiles$nyha[i],
      shock = profiles$shock[i]
    )
    nd$survival <- predict(fit2, newdata = nd, type = "survival") * 100
    nd$profile  <- profiles$label[i]
    nd
  })
  curve_df <- do.call(rbind, curve_list)

  ggplot(curve_df, aes(time, survival, colour = profile)) +
    geom_line() +
    scale_y_continuous(limits = c(0, 100)) +
    labs(x = "Months after surgery",
         y = "Freedom from death (%)",
         title = "Predicted survival by risk profile",
         colour = NULL) +
    theme_minimal()
}
```

```
# -- Multiphase predictions with decomposition --------------------
set.seed(42)
n   <- 200
dat <- data.frame(
  time   = rexp(n, rate = 0.25) + 0.01,
  status = rbinom(n, size = 1, prob = 0.65)
)
fit_mp <- hazard(
  survival::Surv(time, status) ~ 1,
  data   = dat,
  dist   = "multiphase",
  phases = list(
    early = hzr_phase("cdf", t_half = 0.5, nu = 2, m = 0,
                      fixed = "shapes"),
    late  = hzr_phase("cdf", t_half = 5,   nu = 1, m = 0,
                      fixed = "shapes")
  ),
  fit     = TRUE,
  control = list(n_starts = 5, maxit = 1000)
)

t_grid <- seq(0.01, max(dat$time) * 0.9, length.out = 100)
nd     <- data.frame(time = t_grid)

# Overall survival
predict(fit_mp, newdata = nd, type = "survival")

# Per-phase decomposed cumulative hazard
decomp <- predict(fit_mp, newdata = nd,
                  type = "cumulative_hazard", decompose = TRUE)
head(decomp)
```

`predict.hzr_outhaz` *Predictions from a fit loaded out of a SAS* `OUTHAZ=` *dataset*

Description

Rebuilds the multiphase model the `OUTHAZ=` dataset describes (which phases are in it, their shapes, and the fitted parameter vector) and then predicts exactly as `predict.hazard()` does.

Usage

```
## S3 method for class 'hzr_outhaz'
predict(
  object,
  newdata,
```

```
  type = c("hazard", "survival", "cumulative_hazard"),
  decompose = FALSE,
  se.fit = FALSE,
  level = 0.95,
  conf.type = c("log-log", "logit"),
  ...
)
```

Arguments

`object`	An `hzr_outhaz` object from `hzr_read_outhaz()`.
`newdata`	Data frame with a `time` column. Required.
`type`	One of `"hazard"`, `"survival"` or `"cumulative_hazard"`; default `"hazard"`, as in `predict.hazard()`.
`decompose`	Accepted only to keep this method's argument list identical to `predict.hazard()`'s, so that a positional call means the same thing for both. `TRUE` is an error: an `OUTHAZ=` dataset names its phases but carries no per-phase decomposition, and silently returning the total prediction instead would answer a question that was not asked.
`se.fit`	Logical; add delta-method standard errors and confidence limits, as `predict.hazard()` does.
`level`	Numeric confidence level in `(0, 1)`; default `0.95`. Only used when `se.fit = TRUE`. **SAS draws narrower bands than this by default.** `PROC HAZPRED` takes its width from `CLEVEL`, whose default is `0.68268948`, documented in the macro source as "(1 sd)", so its `T_ALPHA` multiplier is 1 to seven decimals (the literal is truncated) and the band is one standard error, 68.3%, not 95%. Reproducing a SAS figure at this function's default therefore yields a band about 1.96 times wider than the one being checked against, with no error and no warning on either side. Pass the SAS level explicitly to match: `predict(fit, newdata, type = "survival", se.fit = TRUE, level = 2 * stats::pnorm(1) - 1, conf.type = "logit")` The default is left at `0.95` deliberately: it is the right R-side default, and silently adopting SAS's would make this method disagree with every other R modelling function.
`conf.type`	Transform for `type = "survival"` confidence limits when `se.fit = TRUE`: `"log-log"` (default) or `"logit"`, which reproduces SAS `PROC HAZPRED`'s survival limits. A real argument rather than part of `...` because a misspelling would otherwise be swallowed silently and return log-log limits that disagree with the SAS job being reproduced. As in `predict.hazard()`, the value is checked only where it is used, so an ignored one does not make a point or hazard prediction fail.
`...`	Must be empty. Anything landing here is an error rather than a silently ignored argument.

Details

newdata is required. An OUTHAZ= dataset holds a converged model and no data, so there is no fitted time vector to fall back on; without newdata the prediction would be over nothing.

Value

What predict.hazard() returns for a fit with no covariates: a numeric vector of predictions when se.fit = FALSE, and when se.fit = TRUE a data frame with columns fit, se.fit, lower and upper (the confidence limits at level). The decomposed long-format return of predict.hazard() is not reachable here, because decompose = TRUE is refused.

What is reconstructed, and what is refused

A phase is in the model when its intercept row (E0, C0, L0) carries _STATUS_ = 1, which is the test PROC HAZPRED itself applies. The early phase becomes hzr_phase("cdf") and the late phase hzr_phase("g3"), with the shape estimates read off the DELTA/THALF/NU/M and TAU/GAMMA/ALPHA/ETA rows.

These cases error rather than return a number that looks like a prediction: a dataset carrying covariates (their coefficients cannot be matched to newdata columns from the file alone); a non-zero DELTA (the early-phase time transformation is not implemented); and a G1FLAG that disagrees with the signs of the M and NU estimates.

With se.fit = TRUE there are three more, because the stored covariance is on SAS's estimation scale and has to be mapped onto this package's:

- a fit constrained by FIXMNU1, which ties M to 1/NU;
- a fit estimating GAMMA, ALPHA or ETA on one of PROC HAZARD's *composite* late-phase scales, log(GAMMA*ETA - 2) or log(GAMMA*ETA/ALPHA - 2) rather than log() of the parameter. This is the ordinary unconstrained late phase, not an exotic case: with G3FLAG = 1 and no FIXGE2/FIXGAE2, ETA and ALPHA are always on a composite scale and GAMMA is whenever ETA is fixed. So an early + constant + late fit with any free late shape gets point predictions but no standard errors;
- a fit under FIXGE2 or FIXGAE2 where a late parameter is *derived* from an estimated one (ETA = 2/GAMMA, GAMMA = 2/ETA, ALPHA = GAMMA*ETA/2). The derived parameter has no covariance row, so its contribution to the variance would be dropped without trace.

See Also

hzr_read_outhaz(), predict.hazard().

Examples

```
f <- system.file("extdata", "outhaz-fixture.rds", package = "TemporalHazard")
if (nzchar(f)) {
  fit <- hzr_read_outhaz(f)
  predict(fit, newdata = data.frame(time = c(1, 6, 12)), type = "survival")
}
```

print.hzr_calibrate *Print method for hzr_calibrate*

Description

Print method for hzr_calibrate

Usage

```
## S3 method for class 'hzr_calibrate'
print(x, digits = 3, ...)
```

Arguments

x	An hzr_calibrate object.
digits	Number of decimal places for formatting.
...	Additional arguments (ignored).

Value

The object x of class c("hzr_calibrate", "data.frame"), invisibly. The data frame has one row per quantile group and columns: group (group index), n (group size), events (event count), mean, min, max (covariate summary within group), prob (observed event probability), link_value (transformed probability on the link scale). When stratified via the by argument, a by column is also present. Attributes: "link" (the transform applied, e.g. "logit") and "groups" (number of quantile bins).

print.hzr_deciles *Print method for hzr_deciles*

Description

Print method for hzr_deciles

Usage

```
## S3 method for class 'hzr_deciles'
print(x, digits = 3, ...)
```

Arguments

x	An hzr_deciles object.
digits	Number of decimal places for formatting.
...	Additional arguments (ignored).

Value

The object x of class c("hzr_deciles", "data.frame"), invisibly. The data frame has one row per risk group and columns: group (integer group index, 1 = lowest risk), n (group size), events (observed event count; weighted for a weighted fit), expected (expected event count from model predictions, net of any entry time, and weighted as events is), observed_rate, expected_rate (events / n; per unit of case weight for a weighted fit, and NA for a group whose weights sum to 0), chi_sq (per-group (O-E)^2/V contribution, where V is the Poisson variance of the weighted count, equal to E for an unweighted fit), p_value (1-df chi-square upper-tail p), mean_survival, mean_cumhaz (mean predicted values in group). An "overall" attribute contains the omnibus chi-square test (fields: chi_sq, df, p_value, time, groups, total_events, total_expected, n_included, n_excluded).

---

print.hzr_gof *Print method for hzr_gof*

Description

Print method for hzr_gof

Usage

```
## S3 method for class 'hzr_gof'
print(x, digits = 3, ...)
```

Arguments

x	An hzr_gof object.
digits	Number of decimal places for formatting.
...	Additional arguments (ignored).

Value

The object x of class c("hzr_gof", "data.frame"), invisibly. The data frame has one row per time point and columns: time, n_risk, n_event, n_censor, km_surv (Kaplan-Meier survival), km_cumhaz, par_surv (parametric survival), par_cumhaz, cum_observed (cumulative observed events), cum_expected (cumulative expected events from model), residual (cum_expected - cum_observed). Multiphase models additionally include par_cumhaz_<phase> columns for per-phase cumulative hazard contributions. A "summary" attribute contains scalar diagnostics: total_observed, total_expected, final_residual, dist, n, n_tallied. When n_tallied is below n, the printed totals cover only those subjects, and a note says so.

print.hzr_kaplan *Print method for hzr_kaplan*

Description

Print method for hzr_kaplan

Usage

```
## S3 method for class 'hzr_kaplan'
print(x, digits = 4, n = 20, ...)
```

Arguments

x	An hzr_kaplan object.
digits	Number of decimal places for formatting.
n	Maximum rows to print (default 20).
...	Additional arguments (ignored).

Value

The object x of class c("hzr_kaplan", "data.frame"), invisibly. The data frame has one row per event time (or all times when event_only = FALSE) and columns: time (follow-up time), n_risk (number at risk), n_event (events in interval), n_censor (censored observations in interval), survival (Kaplan-Meier survival estimate), std_err (Greenwood standard error on log-hazard scale), cl_lower, cl_upper (logit-transformed confidence limits on the survival scale), cumhaz (Nelson-Aalen cumulative hazard), hazard (interval hazard estimate), density (estimated event density), life (life-table life expectancy contribution).

print.hzr_sas_job *Print method for hzr_sas_job*

Description

Shows what hzr_translate_sas() made of one SAS job: the chunks it emitted, how many of the job's tokens it mapped, and one line per construct it could not translate. Read the untranslated lines before trusting the document – each one names something the emitted R does differently from PROC HAZARD, or not at all.

Usage

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

Arguments

x	An hzr_sas_job object, as returned by hzr_translate_sas().
...	Additional arguments (ignored).

Value

The object x, invisibly.

See Also

hzr_translate_sas(), which builds the object.

---

stepwise_trace *Extract the captured console trace from an* hzr_stepwise *fit*

Description

Every run of hzr_stepwise() records the header, per-step lines, and final summary regardless of the trace flag. This accessor returns the full character vector for display or logging.

Usage

```
stepwise_trace(fit)
```

Arguments

fit	An hzr_stepwise object.

Value

Character vector, one element per console line.

See Also

hzr_stepwise(), which produces the object this accessor reads.

Examples

```
data(avc)
avc <- na.omit(avc)
base <- hazard(survival::Surv(int_dead, dead) ~ age,
               data = avc, dist = "weibull", fit = TRUE,
               theta = c(mu = 0.01, nu = 0.5, 0))

sw <- hzr_stepwise(base, scope = ~ age + mal,
                   data = avc, direction = "forward")
cat(stepwise_trace(sw), sep = "\n")
```

summary.hazard *Summarize a hazard model*

Description

Returns a compact summary of a hazard object, including model metadata, fit diagnostics, and coefficient-level statistics when available.

Usage

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

Arguments

object	A hazard object.
...	Unused; for S3 compatibility.

Value

An object of class summary.hazard. Its log_lik is the log-likelihood at the estimates, and NA for a fit with objective = "sas" that read an interval-censored row (one of positive weight, not dropped by a phase design): there the fitted objective is PROC HAZARD's interval-mean-hazard objective, not a log-likelihood. That value is always in objective_value, and objective says which of the two it is ("likelihood" or "sas"). For a Weibull fit whose scale mu, or its variance, cannot be represented, the coefficient table has a log(mu) row under mu, with the standard error of log(mu); it is not tested against 0.

See Also

hazard() for model fitting, predict.hazard() for predictions.

vignette("fitting-hazard-models") for fitting workflows, vignette("inference-diagnostics") for bootstrap CIs and diagnostics.

Examples

```
# -- Single-phase Weibull summary ----------------------------------
set.seed(1)
fit <- hazard(time = rexp(30, 0.5), status = rep(1L, 30),
              theta = c(0.3, 1.0), dist = "weibull", fit = TRUE)
summary(fit)


# -- Multiphase model summary --------------------------------------
set.seed(42)
n   <- 200
dat <- data.frame(
  time   = rexp(n, rate = 0.25) + 0.01,
```

```
  status = rbinom(n, size = 1, prob = 0.65)
)
fit_mp <- hazard(
  survival::Surv(time, status) ~ 1,
  data   = dat,
  dist   = "multiphase",
  phases = list(
    early = hzr_phase("cdf", t_half = 0.5, nu = 2, m = 0,
                      fixed = "shapes"),
    late  = hzr_phase("cdf", t_half = 5,   nu = 1, m = 0,
                      fixed = "shapes")
  ),
  fit     = TRUE,
  control = list(n_starts = 5, maxit = 1000)
)
summary(fit_mp)
```

`tga` *TGA: Transposition of the Great Arteries*

Description

Survival data for 470 patients who underwent the arterial switch operation for transposition of the great arteries at Boston Children's Hospital and Children's Hospital of Philadelphia. Used for sensitivity analysis and internal validation examples.

Usage

```
tga
```

Format

A data frame with 470 rows and 14 variables:

study Patient identifier

simple Simple TGA indicator (0/1)

dextroin D-looped transposition indicator (0/1)

ca_1rl2c Coronary artery pattern indicator

hyaaproc Hybrid approach procedure indicator (0/1)

no_tca No total circulatory arrest indicator (0/1)

tca_time Total circulatory arrest time (minutes)

age_days Age at operation (days)

arciopyr Aortic cross-clamp time per year

dead Death indicator (1 = dead, 0 = censored)

int_dead Follow-up interval to death or last contact (months)

source Source institution (BCH or CHOP)

ca1_2_l Coronary artery configuration (1/2/L)

opyear Year of operation

Source

Boston Children's Hospital and Children's Hospital of Philadelphia.

See Also

Other datasets: avc, cabgkul, omc, uslife2023, valves

`uslife2023` *US Life Table 2023: All-Interval-Censored SAS Parity Anchor*

Description

The published NCHS United States life table for 2023, expressed on a synthetic radix of 100,000, as a `PROC HAZARD` job consumes it: one interval-censored row per year of age, weighted by the number of deaths falling in that year. Aggregate published counts only: no patient-level data and no PHI.

Usage

```
uslife2023
```

Format

A data frame with 124 rows and 3 variables:

age_l Lower bound of the age interval, in years (integer)

age_u Upper bound of the age interval, in years (integer); `age_u - age_l` is exactly 1 on every row

d_all Deaths in the interval on a 100,000 radix. This is the ICENSOR variable, and it is a *weight*, not an indicator (a fact the weighted life table forces and 335 occurrences of `icensor icens_wt=il_dead` across the SAS corpus corroborate). Sums to 100000.0125; ranges from 0.2352 to 3620.335

Details

This is the reference fixture for `objective = "sas"`. It is the cleanest available anchor for that form for two reasons. Every row is interval-censored, so nothing else dilutes the signal; and every interval is exactly one year wide, so $\log(u - l) = 0$ and the width term switches off, isolating the $S(u)\,\Delta\Lambda$ core. It also settles by itself the rival hypothesis that SAS bakes in a constant width divisor: that reading is off by 593,146 log-likelihood units here.

Two rows of the source table (ages 119–120 and 124–125) carry `d_all == 0` and are dropped by the SAS job's own `IF D_ALL=0 THEN DELETE`, leaving 124 of 126. The age grid is therefore not contiguous, which is why the fixture carries explicit `age_l`/`age_u` columns rather than an implied one-year step.

Source

National Center for Health Statistics, United States Life Tables, 2023. Built by `data-raw/make_data.R` from `inst/extdata/uslife2023.csv`, which needs no SAS license to reproduce.

References

The SAS reference fit is `distributions/hz.icall.lst` in that study, whose printed log-likelihood is -410414 with `Number of events conserved` = 100000.

See Also

`hazard` for the `objective` argument this fixture anchors.

Other datasets: `avc`, `cabgkul`, `omc`, `tga`, `valves`

Examples

```
data(uslife2023)

# Every interval is one year wide -- the property that makes this the anchor.
stopifnot(all(uslife2023$age_u - uslife2023$age_l == 1))

# Deaths by age, on the published 100,000 radix.
plot(uslife2023$age_l, uslife2023$d_all, type = "h",
     xlab = "Age (years)", ylab = "Deaths per 100,000",
     main = "US Life Table 2023: deaths by year of age")
```

---

valves *Valves: Primary Heart Valve Replacement*

Description

Data for 1,533 patients who underwent primary heart valve replacement. The largest multivariable example dataset with multiple endpoints including death, prosthetic valve endocarditis (PVE), bioprosthesis degeneration, and reoperation.

Usage

```
valves
```

Format

A data frame with 1533 rows and 19 variables:

age_cop Age at operation (years)

nyha NYHA functional class (1–4)

mitral Mitral valve position indicator (0/1)

double_ Double valve replacement indicator (0/1)

ao_pinc Aortic position, incompetence (0/1)

black Black race indicator (0/1)

i_path Ischemic pathology indicator (0/1)

nve Native valve endocarditis indicator (0/1)

mechvalv Mechanical valve indicator (0/1)

male Male sex indicator (0/1)

int_dead Follow-up interval to death or last contact (months)

dead Death indicator (1 = dead, 0 = censored)

int_pve Follow-up interval to PVE or last contact (months)

pve PVE indicator (1 = PVE, 0 = censored)

bio Bioprosthesis indicator (0/1)

int_rdg Follow-up interval to degeneration or last contact (months)

reop_dg Reoperation for degeneration indicator (0/1)

int_reop Follow-up interval to reoperation or last contact (months)

reop Reoperation indicator (0/1)

Source

Cleveland Clinic Foundation heart valve replacement registry.

See Also

`vignette("fitting-hazard-models")`, `vignette("prediction-visualization")`

Other datasets: `avc`, `cabgkul`, `omc`, `tga`, `uslife2023`

Examples

```
data(valves)
valves_cc <- na.omit(valves)

# Kaplan-Meier for two endpoints
km_death <- survival::survfit(
  survival::Surv(int_dead, dead) ~ 1, data = valves_cc)
km_pve <- survival::survfit(
  survival::Surv(int_pve, pve) ~ 1, data = valves_cc)

plot(km_death, xlab = "Months after valve replacement", ylab = "Survival",
     main = "Valves: Death and PVE endpoints")
lines(km_pve, col = "red")
legend("bottomleft", c("Death", "PVE"), col = c("black", "red"), lty = 1)
```

vcov.hazard *Extract variance-covariance matrix from hazard model*

Description

Returns the estimated variance-covariance matrix of the fitted coefficients.

Usage

```
## S3 method for class 'hazard'
vcov(object, ...)
```

Arguments

object	A hazard object.
...	Unused; for S3 compatibility.

Value

A numeric matrix containing the estimated variance-covariance matrix of the fitted coefficients, with rows and columns named by the coefficient labels (phase-prefixed for multiphase models, e.g. early.x). Rows and columns for parameters held fixed (e.g. fixed shape parameters) are NA because they carry no variance; the finite free-parameter block is still usable. For Conservation-of-Events fits the conserved phase log_mu *normally* carries a variance: it is removed from the optimizer search but the vcov is the full-information matrix at the optimum (the CoE solution is the unconstrained MLE). That recomputation needs a Hessian (analytic, or from **numDeriv** for left- or interval-censored rows) that can be inverted; if it fails the fit warns, records why in degraded_causes, and has no covariance matrix, so vcov() returns NA. For a Weibull fit whose scale mu, or its variance, cannot be represented, mu's row and column are NA, the reason is recorded in the fit's degraded_causes, and summary() shows the variance of log(mu) instead. Returns a scalar NA only when the model has not been fitted or no covariance matrix is available.

Examples

```
set.seed(1)
fit <- hazard(time = rexp(30, 0.5), status = rep(1L, 30),
              theta = c(0.3, 1.0), dist = "weibull", fit = TRUE)
vcov(fit)
```

Index

∗ **datasets**
avc, 3
cabgkul, 4
omc, 65
tga, 79
uslife2023, 80
valves, 81

as.data.frame.hzr_stepwise *(*hzr_stepwise*)*, 52
avc, 3, *5*, *65*, *80–82*

base::subset(), *7*
base::transform(), *7*

cabgkul, *4*, 4, *65*, *80–82*
coef.hazard, 5

hazard, 6, *81*
hazard(), *18*, *21*, *23*, *27*, *31*, *35*, *43–46*, *59–61*, *63*, *69*, *78*
hzr_argument_mapping, 18
hzr_argument_mapping(), *30, 31*
hzr_bootstrap, 19
hzr_bootstrap(), *11*, *14*
hzr_calibrate, 24
hzr_clamp_prob, 25
hzr_clamp_prob(), *40, 41*
hzr_competing_risks, 26
hzr_deciles, 28
hzr_deciles(), *25*, *37*
hzr_decompos, 30
hzr_decompos(), *18*, *32, 33*, *46*, *48, 49*
hzr_decompos_g3, 32
hzr_decompos_g3(), *18*, *30*, *45, 46*
hzr_evaluate, 34
hzr_evaluate(), *10*, *66*
hzr_gof, 35
hzr_gof(), *28*, *39*
hzr_kaplan, 38
hzr_kaplan(), *27*, *42, 43*
hzr_log1mexp, 40
hzr_log1mexp(), *26*, *41*
hzr_log1pexp, 41
hzr_log1pexp(), *26*, *40*
hzr_nelson, 42
hzr_phase, 43
hzr_phase(), *9*, *13*, *15*, *18*, *31*, *59, 60*, *69*
hzr_phase_cumhaz, 47
hzr_phase_cumhaz(), *31*, *33*, *46*, *48, 49*
hzr_phase_hazard, 48
hzr_phase_hazard(), *46*, *48*
hzr_read_outhaz, 50
hzr_read_outhaz(), *72, 73*
hzr_repeated_events, 51
hzr_repeated_events(), *9*, *63*
hzr_stepwise, 52
hzr_stepwise(), *11*, *20–23*, *77*
hzr_theta_names, 60
hzr_theta_names(), *34, 35*
hzr_translate_sas, 61
hzr_translate_sas(), *76, 77*

is_hzr_phase, 64

omc, *4, 5*, 65, *80–82*
optim, *12*

predict(), *15*, *45*
predict.hazard, 65
predict.hazard(), *9*, *15*, *29*, *37*, *61*, *71–73*, *78*
predict.hzr_outhaz, 71
predict.hzr_outhaz(), *50*
print(), *34*
print.hzr_bootstrap *(*hzr_bootstrap*)*, 19
print.hzr_calibrate, 74
print.hzr_competing_risks *(*hzr_competing_risks*)*, 26
print.hzr_deciles, 74

print.hzr_gof, 75
print.hzr_kaplan, 76
print.hzr_nelson *(*hzr_nelson*)*, 42
print.hzr_phase *(*hzr_phase*)*, 43
print.hzr_sas_job, 76
print.hzr_stepwise *(*hzr_stepwise*)*, 52
print.summary.hzr_stepwise
*(*hzr_stepwise*)*, 52

stats::lm(), *7*, *9*
stats::nlm(), *10, 11*, *14*
stats::optim(), *11*
stats::predict.lm(), *68*
stepwise_trace, 77
stepwise_trace(), *59*
summary.hazard, 78
summary.hazard(), *15*, *69*
summary.hzr_stepwise *(*hzr_stepwise*)*, 52
survival::Surv(), *8*
survival::survfit(), *35, 36*

tga, *4, 5*, *65*, 79, *81, 82*

uslife2023, *4, 5*, *65*, *80*, 80, *82*
utils::txtProgressBar(), *20*

valves, *4, 5*, *65*, *80, 81*, 81
vcov(), *45*
vcov.hazard, 83
vcov.hazard(), *23*