Package 'FastSurvival'

May 27, 2026

Type Package

Title Fast Kaplan-Meier, Log-Rank, and Hazard Ratio Estimation for Survival Analysis

Version 0.1.0

Description Provides fast alternatives to standard survival analysis functions in the 'survival' package. The package implements a single-time-point Kaplan-Meier estimator (survfit_fast()), a log-rank test (survdiff_fast()), a closed-form hazard ratio estimator based on the Pike-Halley Estimator method (coxph_fast()), and a clinical trial data simulator (simdata_fast()). All functions are designed for repeated evaluation inside large simulation loops, such as adaptive sample-size re-estimation, probability-of-success calculations, and regional consistency evaluation in multi-regional trials. Core computations are implemented in 'C++' via 'Rcpp' for maximum performance. Methodological background is described in Collett (2014, ISBN:9780429196294).

License MIT + file LICENSE

URL https://github.com/gosukehommaEX/FastSurvival

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

Encoding UTF-8

Language en-US

Depends R (>= 4.1.0)

Imports dqrng, Rcpp

LinkingTo Rcpp, dqrng

Suggests survival, microbenchmark, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

VignetteBuilder knitr

NeedsCompilation yes

Author Gosuke Homma [aut, cre]

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

Repository CRAN

Date/Publication 2026-05-27 08:40:07 UTC

Contents

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

Description

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

Usage

```
coxph_fast(time, event, group, control, conf.int = 0.95, presorted = FALSE)
```

Arguments

`time`	A numeric vector of follow-up times for all subjects (pooled over both groups).
`event`	An integer or numeric vector of event indicators (1 = event, 0 = censored), aligned with `time`.
`group`	A vector of group labels aligned with `time`. Any type that supports equality comparison is accepted.
`control`	A scalar value indicating which level of `group` represents the control group. Subjects with `group != control` are treated as the treatment group.
`conf.int`	A single numeric value in (0, 1) specifying the confidence level for the Wald interval. Defaults to 0.95.
`presorted`	A logical value. If `TRUE`, `time`, `event`, and `group` are assumed to be already sorted in ascending order of `time`, and the internal `order()` call is skipped. If `FALSE` (default), sorting is handled internally.

Details

Let t_k (k = 1, ..., K) denote the distinct observed event times in the pooled sample. At each t_k, let n_T_k and n_C_k be the numbers at risk in the treatment and control groups just before t_k, and let O_T_k and O_C_k be the numbers of events in each group, with n_k = n_T_k + n_C_k and O_k = O_T_k + O_C_k. Define E_T = sum n_T_k O_k / n_k and E_C = sum n_C_k O_k / n_k as the log-rank expected event totals.

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

p_k = n_T_k theta_0 / (n_C_k + n_T_k theta_0) U_0 = sum (O_T_k - O_k p_k) I_0 = sum O_k p_k (1 - p_k) J_0 = sum O_k p_k (1 - p_k) (1 - 2 p_k)

Third, the closed-form Halley correction is applied:

delta_hat = U_0 / I_0 - J_0 U_0^2 / (2 I_0^3) theta_hat = theta_0 exp(delta_hat)

The residual error satisfies |theta_hat - theta_Cox| = O_p(n^{-3/2}), three orders of magnitude faster than the O_p(n^{-1/2}) rate of Peto and Pike, and the per-call cost is approximately thirty times lower than that of the iterative Cox solver (Homma, 2025).

The Wald standard error on the log scale is SE = 1 / sqrt(I_0), where I_0 is the observed information evaluated at the Pike anchor. This is the same quantity used in the Wald confidence interval reported by `summary(coxph(...))`, which is based on the observed information at the maximum likelihood estimate. Because the Pike anchor lies within O_p(n^{-1/2}) of the Cox maximum likelihood estimate, the difference between I_0 and the information at the maximum likelihood estimate is negligible for the purpose of interval construction.

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

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

Value

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

`coef` Log hazard ratio log(theta_hat).

`exp(coef)` Hazard ratio theta_hat (point estimate).

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

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

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

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

References

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

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

See Also

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

Examples

```
library(survival)

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

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

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


library(microbenchmark)
microbenchmark(
  coxph_fast = coxph_fast(ovarian$futime, ovarian$fustat, ovarian$rx, 2),
  coxph      = coxph(Surv(futime, fustat) ~ rx, data = ovarian),
  times = 1000
)
```

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

Description

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

Usage

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

Arguments

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

Value

Invisibly returns x.

See Also

coxph_fast

Examples

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

---

print.survdiff_fast *Print Method for survdiff_fast Objects*

Description

Formats and prints a survdiff_fast object similarly to print(survival::survdiff(...)), showing the observed and expected event counts for the control and treatment groups, the per-group contributions (O-E)^2 / E and (O-E)^2 / V, the test statistic, and the corresponding p-value.

Usage

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

Arguments

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

Value

Invisibly returns `x`.

See Also

`survdiff_fast`

Examples

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

---

print.survfit_fast *Print Method for survfit_fast Objects*

Description

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

Usage

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

Arguments

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

Value

Invisibly returns `x`.

See Also

`survfit_fast`

Examples

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

`simdata_fast` *Fast Survival Data Simulation for Clinical Trials*

Description

Simulates individual patient data for one- or two-group time-to-event trials. Patient accrual times are drawn from a piecewise uniform distribution. Survival and dropout times are drawn from either a simple exponential or a piecewise exponential distribution, depending on whether a scalar or vector hazard is supplied. Random number generation uses `dqrng` for speed. C++ backends handle piecewise sampling and two-group interleaving to minimize R-level overhead.

Usage

```
simdata_fast(
  nsim = 1000,
  n,
  alloc = c(1, 1),
  a.time,
  a.rate,
  e.hazard = NULL,
  e.median = NULL,
  e.time = NULL,
  d.hazard = NULL,
  d.median = NULL,
  d.time = NULL,
  seed = NULL
)
```

Arguments

`nsim`	A positive integer specifying the number of simulation iterations. Default is 1000.
`n`	A positive integer (one group) or a numeric vector of length 2 (two groups: `c(n_control, n_treatment)`) specifying the per-group sample sizes per simulation. Alternatively, supply the total sample size as a scalar together with `alloc` to split automatically.
`alloc`	A numeric vector of length 2 specifying the allocation ratio `c(control, treatment)`. Used only when `n` is a scalar. Default is `c(1, 1)`.

a.time	A numeric vector of accrual interval boundaries c(0, t1, ..., T) with a.time[1] = 0. Must be strictly increasing. The last element is the end of the accrual window.
a.rate	A numeric vector of accrual rates (patients per time unit) for each interval. Length must equal length(a.time) - 1. All values must be positive.
e.hazard	Hazard rate(s) for the event time. For a simple exponential, supply a scalar (one group) or a list of two scalars (two groups). For a piecewise exponential, supply a numeric vector (one group) or a list of two numeric vectors (two groups); e.time must then also be supplied. Cannot be used together with e.median.
e.median	Median survival time(s) for the event time. Same structure as e.hazard. Converted internally via hazard = log(2) / median. Cannot be used together with e.hazard.
e.time	Interval boundaries for a piecewise exponential event time, of the form c(0, t1, ..., Inf). Required when e.hazard or e.median is a vector of length > 1. For two groups, supply a list of two boundary vectors if the change points differ between groups, or a single vector if they are shared.
d.hazard	Hazard rate(s) for the dropout time. Same structure as e.hazard. Set both d.hazard and d.median to NULL to suppress dropout. Cannot be used together with d.median.
d.median	Median dropout time(s). Same structure as e.median. Cannot be used together with d.hazard.
d.time	Interval boundaries for a piecewise exponential dropout time. Same structure as e.time.
seed	A single integer for reproducibility. Passed to dqrng::dqset.seed. Default is NULL (no seed).

Details

For each patient, three times are generated independently:

Accrual time Drawn from a piecewise uniform distribution defined by a.time and a.rate. The probability of falling in interval j is proportional to a.rate[j] * (a.time[j+1] - a.time[j]).

Survival time Drawn from a simple exponential distribution when e.hazard is a scalar (or e.median is a scalar), or from a piecewise exponential distribution when e.hazard is a vector (with e.time required). The inverse-CDF method is used for the piecewise case.

Dropout time Drawn from the same family as survival time, governed by d.hazard / d.median and d.time. Set to Inf when d.hazard and d.median are both NULL (no dropout).

The observed time-to-event is tte = pmin(surv_time, dropout_time). The event indicator is 1 when surv_time <= dropout_time and 0 otherwise. The calendar time is calendar_time = accrual_time + tte.

Exactly one of e.hazard and e.median must be supplied. Exactly one of d.hazard and d.median must be supplied when dropout is modeled; both must be NULL to suppress dropout entirely.

For a two-group trial, group-specific parameters are passed as a list of length 2, where element 1 corresponds to the control group and element 2 to the treatment group. For a one-group trial, plain vectors or scalars are passed directly.

Value

A `data.frame` with `nsim * sum(n)` rows and the following columns:

`sim` Simulation ID (integer, 1 to `nsim`).

`group` Group label (integer: 1 for control, 2 for treatment). Always 1 for one-group simulations.

`accrual_time` Patient accrual time from study start.

`surv_time` Survival time from patient entry.

`dropout_time` Dropout time from patient entry (`Inf` when dropout is not modeled).

`tte` Observed time-to-event: `pmin(surv_time, dropout_time)`.

`event` Event indicator: 1 if `surv_time <= dropout_time`, 0 otherwise.

`calendar_time` Calendar time from study start: `accrual_time + tte`.

Examples

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

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

# Two-group simulation, piecewise exponential (delayed treatment effect)
df3 <- simdata_fast(
  nsim     = 100,
  n        = c(100, 100),
  a.time   = c(0, 12),
  a.rate   = 200 / 12,
  e.hazard = list(c(0.08, 0.08), c(0.08, 0.04)),
  e.time   = c(0, 6, Inf),
  seed     = 3
)
head(df3)
```

```
# Two-group via total n + allocation ratio
df4 <- simdata_fast(
  nsim     = 100,
  n        = 200,
  alloc    = c(1, 1),
  a.time   = c(0, 12),
  a.rate   = 200 / 12,
  e.hazard = list(0.08, 0.05),
  seed     = 4
)
head(df4)
```

`survdiff_fast` *Fast Log-Rank Test for Two-Group Survival Data*

Description

Computes the log-rank test statistic for comparing survival curves between two groups. Returns either a one-sided Z-score or a two-sided chi-square statistic. The C++ backend uses a two-pointer merge scan over pooled sorted vectors, eliminating the `rank()` call that dominates the pure-R implementation.

Usage

```
survdiff_fast(time, event, group, control, side, presorted = FALSE)
```

Arguments

`time`	A numeric vector of follow-up times for all subjects.
`event`	An integer or numeric vector of event indicators (1 = event, 0 = censored), aligned with `time`.
`group`	A vector of group labels aligned with `time`.
`control`	A scalar value indicating which level of `group` represents the control group.
`side`	An integer, either 1 or 2. If `side = 1`, returns the standardized log-rank statistic (Z-score, positive when the treatment group has fewer events than expected). If `side = 2`, returns the chi-square statistic (Z^2).
`presorted`	A logical value. If `TRUE`, `time`, `event`, and `group` are assumed to be sorted in ascending order of `time`, and the internal `order()` call is skipped. If `FALSE` (default), sorting is handled internally.

Details

The log-rank statistic is computed as:

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

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

When `presorted = TRUE`, the input vectors are assumed to be sorted in ascending order of `time` and the internal `order()` call is skipped. When `presorted = FALSE` (default), sorting is handled internally. In simulation loops where the data are generated in sorted order, setting `presorted = TRUE` avoids one O(n log n) pass.

The C++ core (`logrank_core`) walks the pooled sorted data with a single two-pointer scan, maintaining running at-risk counts per group. No rank vector is constructed, so the dominant O(n log n) cost of `rank()` in the pure-R version is removed.

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

Value

An object of class `"survdiff_fast"`, which is a length-one numeric value with attributes `O0`, `E0`, `O1`, `E1`, `V1`, `side`, and `n`. The numeric value is the Z-score when `side = 1`, or the chi-square statistic when `side = 2`. Returns `NA_real_` (still with class `"survdiff_fast"`) when `V1 = 0` (e.g., all events in one group).

References

Mantel, N. (1966). Evaluation of survival data and two new rank order statistics arising in its consideration. *Cancer Chemotherapy Reports*, *50*(3), 163-170.

Peto, R., & Peto, J. (1972). Asymptotically efficient rank invariant test procedures. *Journal of the Royal Statistical Society. Series A (General)*, *135*(2), 185-198.

See Also

`survdiff` for the standard implementation. `print.survdiff_fast` for the print method.

Examples

```
library(survival)

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

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

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

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


library(microbenchmark)
microbenchmark(
  survdiff_fast = survdiff_fast(ovarian$futime, ovarian$fustat,
                                ovarian$rx, 2, side = 2),
  survdiff      = survdiff(Surv(futime, fustat) ~ rx, data = ovarian),
  times = 1000
)
```

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

Description

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

Usage

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

Arguments

`t_sorted`	A numeric vector of event or censoring times. Must be sorted in ascending order when `presorted = TRUE`.
`e_sorted`	An integer or numeric vector of event indicators (1 = event, 0 = censored), aligned with `t_sorted`.
`t_eval`	A single numeric value specifying the time point at which the survival probability is evaluated.
`conf.int`	A single numeric value in (0, 1) specifying the confidence level. Defaults to 0.95.

`conf.type` A character string specifying the confidence interval type. Must be one of `"plain"`, `"log"`, or `"log-log"`. Defaults to `"log"`.

`presorted` A logical value. If `TRUE` (default), `t_sorted` and `e_sorted` are assumed to be sorted in ascending order of time. If `FALSE`, the vectors are sorted internally before computation.

Details

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

The standard error is estimated by Greenwood's formula:

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

where d_i is the number of events and n_i is the number at risk at time t_i. The output field `std.err` follows the convention of `survfit`, which reports SE[S(t)] / S(t) (i.e., the standard error on the log scale) when `conf.type != "plain"`, and SE[S(t)] when `conf.type = "plain"`. This function always returns SE[S(t)] (the standard error on the survival scale).

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

When `presorted = TRUE` (default), `t_sorted` and `e_sorted` are assumed to be sorted in ascending order of time. When `presorted = FALSE`, the vectors are sorted internally before computation.

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

- `"plain"`: Linear interval on the survival scale, S(t) +/- z * SE. The bounds are clipped to [0, 1].
- `"log"`: Interval on the log scale (default in `survfit`), S(t) * exp(+/- z * SE / S(t)).
- `"log-log"`: Interval on the complementary log-log scale, S(t)^exp(+/- z * SE / (S(t) * log(S(t)))).

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

Value

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

References

Kaplan, E. L., & Meier, P. (1958). Nonparametric Estimation from Incomplete Observations. Journal of the American Statistical Association, 53(282), 457–481.

See Also

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

Examples

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

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

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

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

Index