Package: BJM (via r-universe)

October 31, 2026

Title Backward Joint Model for the Dynamic Prediction of Both Time-to-Event and Longitudinal Outcomes

Version 0.3.0

Maintainer Wenhao Li <wenhaoli.jlu@gmail.com>

Description Provides tools to fit joint models of multivariate longitudinal data and time-to-event data for dynamic prediction. It allows the joint prediction of both future time-to-event outcomes and future longitudinal outcomes conditional on survival. The models accommodate irregularly measured longitudinal data and competing risks outcomes. The use of the backward joint model enables fast and efficient computation, especially for applications with large sample sizes and many longitudinal variables. Methods without competing risks are described in Shen and Li (2021) <doi:10.1002/sim.9037>, and their extension to competing risks is described in Li et al. (2026) <doi:10.1007/s10985-026-09725-x>.

License MIT + file LICENSE

URL https://wenhaoli18.github.io/BJM/, https://github.com/wenhaoli18/BJM

BugReports https://github.com/wenhaoli18/BJM/issues

LazyData true

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 3.5.0), survival

Imports nlme, mvtnorm, ggplot2 (>= 3.4.0), Matrix, parallel, graphics

Suggests testthat (>= 3.0.0), knitr, rmarkdown, torch, ordinal, timeROC, riskRegression, prodlim

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Wenhao Li [aut, cre], Liang Li [aut]

Repository https://cran.r-universe.dev

Date/Publication 2026-10-31 12:48:18 UTC

RemoteUrl https://github.com/cran/BJM

RemoteRef HEAD

RemoteSha 0fc5e1fa7e43f96709456c2078e209659f00ad0d

Contents

buildRecurrentEpisodes 2
calibrationPlot 8
checkBandcountConvergence 10
cifPlot 12
cmtPlot 14
fitIntervalBJM 16
imputeLongitudinal 19
longitudinalSub 23
pbc3 26
performancePlot 28
plot.longitudinalSub.BJM 30
plot.predictLongitudinal.BJM 31
plot.simulateTrajectory.BJM 32
plot.survivalSub.BJM 34
poolLongitudinalSub 35
predictLongitudinal 37
predictPlot 41
predictRecurrentCount 45
predictRisk 48
riskPlot 53
simulateTrajectory 56
spaghettiPlot 60
survivalSub 61
survivalTrans 63

Index **65**

`buildRecurrentEpisodes`

Reshape recurrent-event data into gap-time episodes for the backward joint model

---

Description

Experimental. Prepares recurrent-event data for the backward joint model by a gap-time (renewal) reformulation: it splits each subject's event history into *episodes* – the interval from one restart (the previous event, or the discharge from the stay it began) to the next event, with the clock reset to `0` at the start of each – and re-clocks any accompanying longitudinal measurements onto the same per-episode timeline. The result plugs into the existing `survivalSub()`/`longitudinalSub()`/`predictRisk()` pipeline, unchanged, to predict for a subject currently at risk in an open episode whether their *next* event (within a horizon) is a recurrence or a terminal event; `predictRecurrentCount` builds on the same data to predict the *number* of recurrences in a window. Unlike the single-event and competing-risks methods the rest of the package rests on (Shen and Li, 2021; Li et al., 2026 – see `citation("BJM")`), this extension is unpublished, and has been checked only in simulation studies, not on real data (see the "Recurrent events" article on the package website).

Usage

```
buildRecurrentEpisodes(
  data_events,
  data_long = NULL,
  id_variable = "id",
  time_variable = "time",
  status_variable = "status",
  time_variable_long = NULL,
  episode_variable = "episode",
  gap_variable = "gap",
  time_in_episode_variable = "time_in_episode",
  event_any_variable = "event_any",
  event_type_variable = "event_type",
  restart_variable = NULL,
  lookback = 0,
  episode_id_variable = "episode_id"
)
```

Arguments

`data_events`	One row per subject per event-or-censoring occurrence, in calendar time (i.e. not yet re-clocked to gap time – this function does that). Must contain `id_variable`, `time_variable` (numeric, strictly increasing within each subject), and `status_variable` (`0` = censored/end of follow-up, `1` = recurrent event, `2` = terminal event – the same 0/1/2 convention pbc3's `status5` already uses). Any other columns are covariates, carried through to `episode_data` from the row that ends each episode: so a covariate that changes over time must hold, on each row, its value at the *start* of the episode that row ends (see Details), not its value at the event. Subjects are matched across `data_events` and `data_long` by the printed value of their id, so an id that is a number in one table and a factor or character in the other still matches; `id` keeps the type it has in `data_events`.
`data_long`	Optional repeated-measurement data to re-clock onto the same per-episode timeline, in the same format `longitudinalSub()`/`predictRisk()` accept elsewhere

in this package: a single bare data.frame, or a list of data.frames (one per longitudinal outcome). Must contain id_variable and time_variable_long. Rows whose time_variable_long falls outside every episode – before 0 (beyond lookback), in a hospital stay lookback does not reach, or after the subject's process has already ended (per data_events) – are dropped, with a message reporting how many. A row whose time_variable_long falls exactly on the shared boundary between two episodes (e.g. a biomarker drawn at the same visit an event is recorded) is genuinely ambiguous – it is at once "the last measurement of the ending episode" and "the first (time_in_episode = 0) measurement of the next one" – so it is kept *twice*, once per episode, rather than silently assigned to only one (which would systematically discard every fresh-episode baseline measurement that happens to coincide with the previous episode's event time, and bias the fitted backward-covariate effect); a message says how many rows this applies to.

id_variable, time_variable, status_variable
: Column names in data_events (defaults "id", "time", "status").

time_variable_long
: Column name of the measurement time in data_long; defaults to time_variable (i.e. the same name as in data_events) when NULL.

episode_variable, gap_variable, time_in_episode_variable, event_any_variable, event_type_variable, episode_id_variable
: Column names to create in the output (defaults "episode", "gap", "time_in_episode", "event_any", "event_type", "episode_id"). episode_id is "<id>:<episode>" (a factor), one value per episode: the grouping variable to use for the biomarker model's random effects (see Details). Change these if they collide with an existing column of data_events/data_long – collisions are an error, not a silent overwrite.

restart_variable
: Optional name of a numeric column of data_events: for each recurrence (status == 1) row, the time the process restarts after it (for hospitalizations, the discharge ending that stay; must not be earlier than the row's time_variable). The next episode starts there instead of at the event time. NULL (default) restarts at the event time itself. A missing value on a recurrence row that starts another episode is taken as no stay, with a warning; a column with no value at all is allowed. A death or the end of follow-up at or before the restart time happened during the stay: see Details. Ignored on rows that end the process. Not carried into episode_data.

lookback
: A single non-negative number (Inf allowed): how far before an episode's start its measurements are still kept, as negative time_in_episode, but never from before the previous event. Default 0 keeps only measurements from the start on.

Details

What counts as an episode. Episode 1 runs from time 0 to a subject's first row in data_events; episode k > 1 runs from the end of episode k - 1 to that subject's k-th row. A subject contributes no further episodes after a row with status %in% c(0, 2) (administrative censoring or a terminal event); any later rows for that id are ignored, with a message saying how many. status == 1 rows (recurrences) always start a new episode.

Feeding the result into `survivalSub()`**.** Fit the gap-time recurrence hazard and the recurrence-vs-terminal-event type model exactly as `survivalSub()` already fits a marginal hazard and a competing-risks type model, using `episode_data` in place of one-row-per-subject data, `Surv(gap, event_any) ~ covariates` in place of `form_marginal_surv`, and `event_type ~ gap + covariates` in place of `form_conditional_cr` (`event_type` is `0` for a recurrence, `1` for a terminal event – so `predictRisk()`'s `risk_prob_1`/`risk_prob_2` come out as the recurrence/terminal risk respectively). Administratively-censored episodes get `event_any = 0` and `event_type = NA`, exactly like `status3 = 0`/`status4 = NA` for a censored subject in `pbc3`. The episodes of one subject are then treated as independent: the coefficients are still consistent for the gap-time model, but the standard errors that `summary()` reports ignore the correlation between a subject's episodes and are too small.

Feeding the result into `longitudinalSub()`**/**`predictRisk()`**.** `gap` plays the same dual role `years` plays for `pbc3`: it is both the `Surv()` time in `episode_data` and the backward covariate swept over candidate values inside `predictRisk()`, so it must appear in `long_sub_fixed` (e.g. `biomarker ~ time_in_episode + gap + gap * time_in_episode + covariates`), and the training data passed to `longitudinalSub()` must be filtered to `event_any == 1` rows of `long_data` – only completed episodes have an observed `gap` – exactly as the package's own examples filter `pbc3` to `status3 == 1` before fitting `longitudinalSub()`. `time_in_episode` (elapsed time since the current episode began) plays the role `year` plays for `pbc3`. To predict from "now" for a subject currently at risk in an open episode, add a trailing `status = 0` row to `data_events` at the current data-cutoff time, and call `predictRisk()` with `prediction_time` set to that subject's elapsed `time_in_episode` at the cutoff.

Random effects: group by episode, not by patient. Fit the biomarker model with the random effects grouped by `episode_id` (`random = ~ time_in_episode | episode_id`), so that each episode is its own "subject", as it already is in the survival sub-model. A patient's biomarker level generally differs from one episode to the next (the underlying trajectory keeps moving), so a single random effect per patient is the wrong unit; in a simulation in which the biomarker's current level drives the risk, grouping by patient gave a clearly worse AUC than grouping by episode. `predictRisk()` and `predictRecurrentCount()` then take one `episode_id` per prediction, so give each prediction its own `id`.

The open episode's `gap` **is** `NA`**.** For an episode that has not ended (`event_any == 0`) – typically the one a subject is currently at risk in – the `gap` column of `long_data` is `NA`: its final length is exactly what is being predicted. `predictRisk()` reads a missing survival-time variable as "event time not yet known, still at risk", so that episode's rows can be passed straight to it, with no placeholder to set or override (`episode_data` keeps the real elapsed time of such an episode, which is what the Cox fit needs as a censoring time). Training rows are unaffected, since `longitudinalSub()` is fit on `event_any == 1` rows only. Note that `predictRisk()` also expects the outcome columns it was fit with (the time and event-type variables) to exist in the prediction data; `long_data` already carries `event_type` (`NA` for an open episode).

Episodes that start at discharge (`restart_variable`**,** `lookback`**).** By default an episode starts at the previous event (time `0` for the first). For hospitalization data that is not the at-risk clock: after a readmission the patient is in hospital, not at risk, until discharge. Give `data_events` a numeric column – passed as `restart_variable` – holding, for each recurrence row, the time the process restarts (the discharge ending that stay); the next episode then starts there and its gap is measured from it. Time `0` is the first episode's start (e.g. the index discharge). Biomarkers measured during the stay, or before the index discharge, precede the episode they predict: `lookback` (default `0`, none) keeps measurements up to that long before an episode's start – never reaching back past the previous event, so a stay's own labs are not mixed into the earlier at-risk period – and gives them *negative*

time_in_episode. Use Inf to keep the whole stay (and, for the first episode, everything before time 0). Negative times are fine for longitudinalSub() and predictRisk().

Death, or the end of follow-up, during a hospital stay. With restart_variable, a terminal event or censoring row whose time is at or before the restart time of the readmission before it happened while the patient was in hospital, not at risk. That readmission then ends the subject's process: no further episode is opened, the row is dropped with a message, and so a death in hospital is not counted as an out-of-hospital terminal event (predictRecurrentCount()'s p_death is the probability of dying after discharge). Record such a stay's restart time as the time of death or of the end of follow-up, or later, when the discharge never happened. A data cutoff during a stay gives no open episode to predict for.

Covariates that change over time. Each episode's covariates come from the row of data_events that ends it, as in counting-process data, where each interval carries the covariate values in force during it. A covariate that changes over time must therefore be recorded on each row as its value at the start of that row's episode (age at the discharge that started it, the number of earlier admissions, the length of the previous stay), and the open episode's row in the same way. A value measured at the event itself – age at readmission, say – depends on how long the episode lasted, which is what the survival sub-model explains, and so biases it; a value measured at the time of the prediction is not what the model was fit with. Biomarkers that change within an episode belong in data_long. Covariates built from earlier episodes can also be added to episode_data afterwards, e.g. the length of the previous gap:

```
ed <- episodes$episode_data
ed$previous_gap <- ave(ed$gap, ed$id, FUN = function(g) c(0, g[-length(g)]))
```

Value

A list with elements:

episode_data One row per subject per episode: id_variable, episode_variable (episode number, starting at 1), gap_variable (that episode's gap time), event_any_variable (0/1: did the episode end in an event of either type, i.e. is gap an observed event time rather than an administrative censoring time), event_type_variable (0 = recurrence, 1 = terminal event, NA when event_any == 0), and every other column of data_events.

long_data NULL if data_long was not supplied; otherwise data_long, in the same bare-data.frame-or-list shape it was supplied in, with episode_variable, gap_variable, time_in_episode_variable, event_any_variable, and event_type_variable columns added (and out-of-range rows dropped, as described above). Any other column of episode_data (baseline or event-time-varying covariates, e.g. age/sex) is also carried onto matching rows here, the way every row of pbc3's long-format table already repeats age/sex/years/status3 – but only into a column data_long does not already have of that name.

Examples

```
## A minimal example: reshaping alone (no model fitting).
## Subject 1: two recurrences, then a terminal event; subject 2: censored.
data_events <- data.frame(
  id     = c(1, 1, 1, 2),
  time   = c(1.2, 2.5, 4.0, 3.1),
  status = c(1,   1,   2,   0),
```

```
  age    = c(58,  58,  58,  61)
)
episodes <- buildRecurrentEpisodes(data_events)
episodes$episode_data

## Hospitalizations: each episode starts at DISCHARGE, not at the readmission.
## `time` = readmission (or death/censoring) day; `discharge` = day the stay
## it started ends. Labs drawn in the stay (and before the index discharge at
## day 0) are kept with negative time_in_episode via lookback = Inf.
events <- data.frame(id = 1, time = c(20, 60, 90), status = c(1, 1, 0),
                     discharge = c(26, 66, NA))
labs <- data.frame(id = 1, day = c(-3, 0, 5, 21, 26, 40, 61, 66, 80))
hf <- buildRecurrentEpisodes(events, labs, time_variable_long = "day",
                             restart_variable = "discharge", lookback = Inf)
hf$episode_data   # gaps measured from each discharge: 20, 34, 24
hf$long_data[, c("day", "episode", "time_in_episode", "gap")]


## A full pipeline on simulated recurrent-event + longitudinal data:
## build episodes -> survivalSub() -> longitudinalSub() -> predictRisk() and
## predictRecurrentCount().
set.seed(1)
n <- 60
data_events <- do.call(rbind, lapply(seq_len(n), function(id) {
  k <- sample(1:3, 1)
  gaps <- rexp(k, rate = 1)
  data.frame(
    id = id, time = cumsum(gaps),
    status = c(rep(1, k - 1), sample(c(0, 2), 1, prob = c(0.6, 0.4))),
    age = round(50 + stats::rnorm(1, sd = 8)), sex = id %% 2
  )
}))

## subject 9999 is still at risk right now, 0.7 time units into their
## 2nd episode (1 prior recurrence, no resolution yet) -- this is who
## we will predict for below.
data_events <- rbind(data_events, data.frame(
  id = 9999, time = c(0.8, 1.6), status = c(1, 0), age = 55, sex = 1
))

data_long <- do.call(rbind, lapply(c(seq_len(n), 9999), function(id) {
  maxt <- max(data_events$time[data_events$id == id]) + 0.2
  tt <- seq(0, maxt, by = 0.25)
  data.frame(id = id, year = tt, biomarker = rnorm(length(tt)))
}))

built <- buildRecurrentEpisodes(data_events, data_long, time_variable_long = "year")

## gap-time recurrence hazard + recurrence-vs-terminal-event type model
survival_fit_recur <- survivalSub(
  built$episode_data,
  Surv(gap, event_any) ~ age + sex + episode,
  event_type ~ gap + age + sex + episode
```

```
)

## longitudinal sub-model, trained on completed episodes only, with the
## random effects grouped by episode (see Details)
long_fit_recur <- longitudinalSub(
  built$long_data[built$long_data$event_any == 1, ],
  biomarker ~ time_in_episode + age + sex + gap + gap * time_in_episode,
  ~ time_in_episode | episode_id
)

## subject 9999's open episode: its gap is NA, which predictRisk() reads as
## "still at risk" -- nothing to override
open <- built$long_data[built$long_data$id == 9999 & built$long_data$event_any == 0, ]
landmark <- max(open$time_in_episode)

## the risk that their *next* event, within 1 time unit, is a recurrence /
## a terminal event
risk <- predictRisk(
  open, long_fit_recur, survival_fit_recur,
  prediction_time = landmark, horizon = 1, time_variable = "time_in_episode",
  survival_variable_all = list(), survival_trans_function = list(),
  bandcount1 = 10, bandcount2 = 10
)
risk$risk_prob_1 # P(next event within horizon is a recurrence)
risk$risk_prob_2 # P(next event within horizon is the terminal event)

## the number of recurrences over the next 3 time units
predictRecurrentCount(
  open, long_fit_recur, survival_fit_recur,
  prediction_time = landmark, horizon = 3, time_variable = "time_in_episode",
  n_sim = 100, seed = 1
)
```

`calibrationPlot` *Plot calibration of dynamic risk predictions*

Description

Checks whether the predicted risks are numerically right, not just well ranked. At each landmark `s` in `prediction_time`, every subject in `data_predict_all` still event-free at `s` gets a predicted risk from `predictRisk` of an event in `(s, s + horizon]`, using only their longitudinal history up to `s`. The subjects are split into `n_groups` groups of about equal size by predicted risk, and for each group the mean predicted risk (horizontal axis) is plotted against the observed risk (vertical axis): one minus the Kaplan–Meier estimate at `s + horizon` or, with competing risks, the Aalen–Johansen cumulative incidence of that event type, both of which account for censoring, with 95\ intervals. Points on the diagonal mean the predictions are well calibrated; points below it mean the model overestimates the risk, above it that it underestimates it.

Evaluating on the data the sub-models were fit on gives an optimistic (apparent) calibration; pass held-out validation data in `data_predict_all` when available.

For an **interval-censored** fit (experimental), the observed risk of a group is a nonparametric (Turnbull-type) estimate from the subjects' intervals, left-truncated at each subject's last visit up to the landmark – with competing risks, the cumulative incidence estimate of Hudgens, Satten and Longini (2001) – so it estimates the same quantity `predictRisk()` predicts, without using the model. Within an interval the timing of the event is not identified, so the estimate is less precise than with exact times, and no confidence interval is drawn. In simulations it was unbiased when the visits were at most about `horizon` apart, but with visits further apart than `horizon` the window mostly falls inside the subjects' intervals and the observed risk of the higher-risk groups was much too high (up to +0.27 with visits every 2–6 years and a 2-year window). An observed risk computed from the model's own posterior was more precise, but followed the predictions of a deliberately miscalibrated model instead of the truth, so it is not offered.

Usage

```
calibrationPlot(
  data_predict_all,
  long_fit_all,
  survival_fit_all,
  prediction_time,
  horizon,
  time_variable,
  survival_variable_all,
  survival_trans_function,
  n_groups = 10,
  bandcount1 = "auto",
  bandcount2 = "auto"
)
```

Arguments

`data_predict_all`
: The evaluation data, in the same format as for `predictRisk`: a list of long-format `data.frames`, one per longitudinal outcome (or a single `data.frame` used for all). Unlike for `predictRisk()`, it must also contain each subject's observed outcome: the survival time and status variables of `survival_fit_all`'s Cox formula and, with competing risks, the event type variable of its `form_conditional_cr`. The outcome is taken from the first data frame and is hidden from `predictRisk()`.

`long_fit_all`
: A `longitudinalSub.BJM` object from `longitudinalSub`.

`survival_fit_all`
: A `survivalSub.BJM` object from `survivalSub`.

`prediction_time`
: A vector of landmark times.

`horizon`
: The prediction horizon (a single positive number): the window after each landmark in which events are counted.

`time_variable`
: The name of the time variable in the linear mixed models.

survival_variable_all, survival_trans_function
: As for predictRisk.

n_groups
: Number of risk groups per landmark. Default 10 (deciles); fewer groups give more stable observed risks in small data.

bandcount1, bandcount2
: As for predictRisk; passed to it at each landmark. "auto" (the default) retunes them in every call, which over a whole cohort is several times slower than fixed values (about 10 times for 160 patients in pbc3): for a large evaluation, choose them once on a few patients with checkBandcountConvergence and pass the numbers.

Value

A ggplot object with one panel per landmark (and, with competing risks, per event type). Its data element is a data.frame with one row per risk group, and columns landmark, cause, group, predicted, observed, lower, upper, n (subjects in the group) and n_cases (events of that type observed in the window).

Examples

```
data(pbc3)
survival_fit_all <- survivalSub(pbc3[!duplicated(pbc3$id), ],
                                Surv(years, status3) ~ age + sex, NULL)
long_sub_fixed <- list("long1" = serBilir ~ year + age + sex + years)
long_sub_random <- list("long1" = ~ year | id)
long_fit_all <- longitudinalSub(list(pbc3[pbc3$status3 == 1, ]),
                                long_sub_fixed, long_sub_random)
trans <- survivalTrans(c(1, 3, 5, 7))

# apparent calibration on the fitting data: 3-year landmark, 2-year window
calibrationPlot(pbc3, long_fit_all, survival_fit_all,
  prediction_time = 3, horizon = 2, time_variable = "year",
  trans$survival_variable_all, trans$survival_trans_function, n_groups = 5)
```

---

checkBandcountConvergence

Check whether bandcount1/bandcount2/bandcount3 are large enough

---

Description

predictRisk()/predictLongitudinal() approximate the integrals behind the predicted risk probabilities (and, for predictLongitudinal(), the predicted biomarker density) with a finite grid, controlled by bandcount1/bandcount2/ bandcount3. There is no universal correct value: too few grid points silently bias the answer, and too many just cost more time, and the right value depends on the data (e.g. how wide the follow-up range is). Rather than guess, or auto-loop until some tolerance is met – which multiplies runtime unpredictably, especially for predictLongitudinal()'s

nested bandcount2 x bandcount3 grid – this runs the prediction once at the bandcount value(s) you supply, once more with those value(s) scaled up, and reports the largest relative change between the two, so you can see whether you have already converged or need to increase the checked bandcount(s) and re-run. It costs exactly 2 calls to predict_fun, regardless of how many times you invoke it.

Usage

```
checkBandcountConvergence(
  predict_fun,
  ...,
  bandcount_args,
  multiplier = 2,
  tol = 0.01
)
```

Arguments

predict_fun	The prediction function to check: predictRisk or predictLongitudinal themselves (not a string, and not predictPlot()/riskPlot(), which return a plot rather than the underlying numeric predictions this function compares). When passing predictLongitudinal, forward a bio_i (via ...) that names exactly **one** biomarker – this function only supports comparing a single biomarker's prediction at a time (see ...).
...	Arguments to forward to predict_fun, exactly as you would call it directly, except for the bandcount argument(s) being checked, which are supplied separately via bandcount_args. Any bandcount argument of predict_fun not named in bandcount_args must still be supplied here (it is held fixed at that value for both calls).
bandcount_args	A named list giving the bandcount value(s) to check, e.g. list(bandcount1 = 10, bandcount2 = 40). Every element is scaled by multiplier for the second call. To isolate which bandcount is driving instability, check one at a time (e.g. list(bandcount2 = 40), with bandcount3 passed as a fixed value via ...), rather than checking all of them together.
multiplier	Factor the checked bandcount value(s) are scaled by for the second call. Must be greater than 1. Defaults to 2 (doubling).
tol	Relative-change tolerance below which the result is reported as converged. Defaults to 0.01 (1%).

Value

An object of class "checkBandcountConvergence.BJM", a named list with elements:

base_bandcount The bandcount value(s) checked, as supplied in bandcount_args.

scaled_bandcount base_bandcount scaled by multiplier.

tol The relative-change tolerance used to decide converged.

by_field A named numeric vector giving the largest relative change, per comparable output field (e.g. risk_prob_1, Y_predict), between the base and scaled call.

max_rel_diff The largest value in by_field, or NA if there was no comparable output (e.g. horizon <= 0).

converged TRUE if max_rel_diff < tol, FALSE if not, NA if there was nothing to compare.

base_result The full result of calling predict_fun at base_bandcount.

scaled_result The full result of calling predict_fun at scaled_bandcount.

Printing the object summarizes these fields.

Examples

```
data(pbc3)

data_survival_fitting = pbc3[!duplicated(pbc3$id), ]
survival_fit_all = survivalSub(data_survival_fitting, Surv(years, status3) ~ age + sex, NULL)

long_sub_fixed = serBilir ~ year + age + sex + (years) + (years) * year
long_sub_random = ~ year | id
long_fit_all = longitudinalSub(pbc3[pbc3$status3 == 1, ], long_sub_fixed, long_sub_random)

trans = survivalTrans(c(1, 3, 5, 7))

data_predict_all = pbc3[pbc3$id == 2 & pbc3$year <= 3, ]

# Check bandcount1/bandcount2 together: are they both already large enough?
check = checkBandcountConvergence(
  predictRisk, data_predict_all, long_fit_all, survival_fit_all,
  prediction_time = 3, horizon = 3, time_variable = "year",
  trans$survival_variable_all, trans$survival_trans_function,
  bandcount_args = list(bandcount1 = 10, bandcount2 = 10)
)
print(check)
```

---

cifPlot *Plot cumulative incidence functions*

Description

Plots the nonparametric cumulative incidence of each event type over time, estimated by the Aalen–Johansen estimator (via survfit on a multi-state Surv outcome). With a single event type this equals one minus the Kaplan–Meier survival curve; with competing risks each event type gets its own curve, correctly accounting for the other event types instead of treating them as censoring. An optional group_variable draws separate curves per group (e.g. treatment arm).

Usage

```
cifPlot(
  data_plot_all,
  survival_variable,
  event_type_variable,
  id_variable = "id",
  censor_value = 0,
  event_labels = NULL,
  group_variable = NULL,
  conf_int = TRUE
)
```

Arguments

data_plot_all	A data.frame with the time-to-event outcome. It may be in long format (several rows per subject, as for the longitudinal sub-model); only the first row of each subject is used, and survival_variable, event_type_variable and group_variable must be constant within each subject.
survival_variable	Name of the time-to-event variable.
event_type_variable	Name of the variable holding each subject's event type, or censoring status.
id_variable	Name of the subject ID column. Default is "id".
censor_value	The value of event_type_variable that marks a censored subject. Default 0. Use NA when censored subjects have a missing event type, as with pbc3$status4. Every other value is treated as an event type.
event_labels	Optional legend labels for the event types: either a character vector named by the event type values (e.g. c("1" = "Death", "2" = "Transplant")), or an unnamed one in the sorted order of the event type values. Default uses the values themselves.
group_variable	Optional name of a subject-level variable to stratify the curves by, shown by line type (and also by color when there is a single event type). NULL (default) for no grouping.
conf_int	Logical; draw pointwise 95\ TRUE.

Value

A ggplot object.

Examples

```
data(pbc3)

# competing risks: death (1) and transplantation (2); 0 = censored
cifPlot(pbc3, survival_variable = "years", event_type_variable = "status5",
  event_labels = c("1" = "Death", "2" = "Transplant"))
```

```
# the same events coded as in status4 (censored subjects are NA)
cifPlot(pbc3, survival_variable = "years", event_type_variable = "status4",
  censor_value = NA, event_labels = c("0" = "Death", "1" = "Transplant"))

# single event type, by treatment arm
cifPlot(pbc3, survival_variable = "years", event_type_variable = "status2",
  group_variable = "drug", conf_int = FALSE)
```

`cmtPlot` *Plot conditional mean trajectories (CMT)*

Description

This function generates the Conditional Mean Trajectories (CMT) plot: the mean biomarker trajectory of the patients whose event happened in the window `(condi_time2event - event_window, condi_time2event]`. The time from the first measurement up to `condi_time2event` is cut into consecutive intervals of length `interval_time`; each point is the mean of those patients' biomarker values measured within one interval, plotted at the interval's end, and a straight line is fitted through the points. With competing risks there is one trajectory per event type.

All times – `time_variable`, `survival_variable`, `condi_time2event`, `event_window` and `interval_time` – are in the units of the data (years, months, days, ...), and the time axis shows `time_variable` on that scale, labelled with its name.

Only patients with an observed event enter: with competing risks, those whose `event_type_variable` equals one of `event_type`; without, those whose `status_variable` is not 0. Without competing risks and with `status_variable = NULL`, patients are selected by their survival time alone, so censored ones are included too, with a warning. An interval without any measurement is left out rather than filled in.

Usage

```
cmtPlot(
  data_plot_all,
  condi_time2event,
  event_type_variable,
  event_type,
  bio_variable,
  time_variable,
  survival_variable,
  interval_time = 1/12,
  id_variable = "id",
  status_variable = NULL,
  event_window = 1
)
```

Arguments

data_plot_all	A data.frame that includes the biomarker used for plotting. It is utilized to plot conditional mean trajectories (CMT).
condi_time2event	Event time the patients are selected around. If NULL, it defaults to the midpoint of the observed range of time_variable in data_plot_all.
event_type_variable	Competing risks variable indicator name. Set to NULL if there are no competing risks.
event_type	A vector containing the values of event_type_variable for the event types to plot (at least two).
bio_variable	Name of the biomarker variable used for plotting.
time_variable	The name of time variable in linear mixed model.
survival_variable	Name of the time-to-event outcomes variable.
interval_time	Length of the intervals the follow-up is cut into, in the units of time_variable.
id_variable	Name of the patient ID column in data_plot_all. Default is "id".
status_variable	Without competing risks, the name of the event indicator (0 = censored), so that only patients with an event are used. Ignored with competing risks.
event_window	Width of the window of event times before condi_time2event that a patient's event must fall in, in the units of survival_variable. Default 1 (one year for data in years; use e.g. 12 for months or 365 for days).

Value

Conditional mean trajectories plot.

Examples

```
# example without competing risks

data(pbc3)

pbc.cmt <- cmtPlot(data_plot_all = pbc3, condi_time2event = 5,
   event_type_variable = NULL, event_type = NULL,
   bio_variable = "serBilir", time_variable = "year",
   survival_variable = "years",
   interval_time = 1/12, status_variable = "status3"
)

pbc.cmt


# example with competing risks

data(pbc3)
```

```
pbc.cmt.cr <- cmtPlot(pbc3, condi_time2event = 5,
   event_type_variable = 'status4', event_type = c("0", "1"),
   bio_variable = "albumin", time_variable = "year",
   survival_variable = "years",
   interval_time = 1/4
)

pbc.cmt.cr
```

`fitIntervalBJM` *Fit the backward joint model to interval-censored event times*

Description

Experimental. With interval censoring nobody's event time T is known exactly – only $L < T \leq R$ – so the longitudinal sub-model $f(Y \mid T)$, which has T as a covariate, cannot be fit on the subjects with an observed event the way it is for right-censored data. `fitIntervalBJM()` fills in T by stochastic EM, then by multiple imputation:

1. fit the marginal survival model to the intervals (`survivalSub()` with `Surv(L, R, type = "interval2")`); it does not depend on T, so it is fit once;
2. start every interval-censored T at its interval midpoint and fit `longitudinalSub()`;
3. draw each subject's T from $f(T \mid L < T \leq R, Y, X) \propto f(Y \mid T, X) f(T \mid X)$ on $(L, R]$ (`imputeEventTime()`), refit `longitudinalSub()`, and repeat for `n_burnin` iterations;
4. keep iterating, saving every `n_thin`-th draw and its `longitudinalSub()` fit until `n_imputations` are saved.

Step 3 uses the backward model's own factorization: $f(Y \mid T)$ on a grid of T is exactly what dynamic prediction already computes (`conditionalYT()`), so each draw costs about one `predictRisk()` call per subject.

For a subject whose event was detected, only the longitudinal measurements up to the last visit without the event (L) are used: the one at the detecting visit R (and any later one) is taken after the event, which $f(Y \mid T)$ does not describe, and they are left out with a message. As in the rest of the interval-censoring support, the biomarker visits are taken to be the inspection visits, so no measurement lies between L and R.

By default only subjects whose event was observed (finite R) enter the longitudinal fit, mirroring the complete-case fit used for right-censored data, which is valid when censoring is independent of (T, Y) given the covariates. `include_right_censored = TRUE` also imputes $T \in (L, \infty)$ for right-censored subjects and uses them; their draws then rely on the survival model's extrapolated tail.

The saved draws come from one chain, so successive draws are correlated; saving only every `n_thin`-th iteration makes them closer to independent. They are drawn with the model parameters at their point estimates, not from their posterior, so the between-imputation variance in `pooled` can still be somewhat understated.

For prediction, `long_fit_all` combines the imputations: its fixed effects are the Rubin's-rules pooled estimates and its residual and random-effects variances the averages over the imputations (see `poolLongitudinalSub`), so a prediction does not depend on which single imputation happened to come first. With competing risks the event-type model of `survival_fit_all` likewise has its coefficients averaged over the imputations.

Usage

```
fitIntervalBJM(
  data_survival,
  data_fit_all,
  form_marginal_surv,
  event_time,
  long_sub_fixed,
  long_sub_random,
  time_variable,
  survival_variable_all = NULL,
  survival_trans_function = NULL,
  form_conditional_cr = NULL,
  n_burnin = 10,
  n_imputations = 5,
  n_thin = 5,
  n_grid = 40,
  baseline = c("spline", "piecewise"),
  df = NULL,
  include_right_censored = FALSE,
  seed = NULL
)
```

Arguments

`data_survival`	One row per subject: the interval endpoints, the survival covariates and the subject id.
`data_fit_all`	As in `longitudinalSub`: one data frame per biomarker (or one shared data frame), long format, for every subject. The `event_time` column and the columns named in `survival_variable_all` are (re)filled from the imputed T.
`form_marginal_surv`	`Surv(L, R, type = "interval2") ~ covariates`.
`event_time`	Name of the event-time column used in `long_sub_fixed`.
`long_sub_fixed, long_sub_random`	As in `longitudinalSub`.
`time_variable`	Name of the visit-time column.
`survival_variable_all, survival_trans_function`	Transformed event-time columns, as in `predictRisk` (`NULL` if none).
`form_conditional_cr`	Optional competing-risks (event-type) model, as in `survivalSub`, e.g. `type ~ Tev + x`. The event type (0/1, in `data_survival` and in every `data_fit_all`

	data frame, alongside the covariates of this formula) is taken as known once the event is detected; each subject's T is then drawn from $f(Y \mid T, D)P(D \mid T)f(T)$, and this model, which uses T, is refit on every draw.
`n_burnin`	Stochastic-EM iterations before draws are saved.
`n_imputations`	Number of saved draws (and longitudinal fits).
`n_thin`	Iterations between saved draws (1 saves every iteration after the burn-in). Larger values make the saved draws less correlated at the cost of `n_burnin + n_imputations * n_thin` iterations in all.
`n_grid`	Grid points per subject interval.
`baseline, df`	Baseline hazard of the survival model (see `survivalSub`).
`include_right_censored`	See Description.
`seed`	Optional integer seed (the global RNG state is restored).

Value

An object of class `"fitIntervalBJM"`, a list with `survival_fit_all` (with competing risks, its event-type model has the coefficients averaged over the imputations, while the rest of that `glm` object, e.g. its standard errors, is the first imputation's fit; `survival_fit_all_list` has one fit per imputation); `long_fit_all_list` (one `longitudinalSub()` fit per imputation); `long_fit_all` (the imputations combined, see Description, for use with `predictRisk()` and friends); `pooled` (`poolLongitudinalSub` of the fits, when `n_imputations > 1`); `imputed_T` (a subjects x imputations matrix); `trace` (each iteration's fixed effects, to check the chain has settled); and `settings` (the iteration settings). `print()` shows a short summary.

Examples

```
# simulated study: visits every 1-3 years, an event known only to lie
# between the last visit without it (L) and the first visit with it (R)
set.seed(1)
n <- 60
x <- rnorm(n)
event_time <- (-log(runif(n)) / (0.05 * exp(0.5 * x)))^(1 / 1.5)
surv <- data.frame(id = seq_len(n), x = x, L = NA_real_, R = NA_real_)
long <- list()
for (i in seq_len(n)) {
  visits <- c(0, cumsum(runif(30, 1, 3)))
  visits <- visits[visits < 12]
  if (event_time[i] > max(visits)) {
    surv$L[i] <- max(visits)
    seen <- visits
  } else {
    k <- findInterval(event_time[i], visits)
    surv$L[i] <- visits[k]
    surv$R[i] <- visits[k + 1]
    seen <- visits[visits <= event_time[i]]
  }
  long[[i]] <- data.frame(id = i, year = seen, x = x[i],
                          y = 1 + 0.3 * seen - 0.2 * event_time[i] + 0.4 * x[i] +
```

```
                            rnorm(1, 0, 0.5) + rnorm(length(seen), 0, 0.3))
}
long <- do.call(rbind, long)

fit <- fitIntervalBJM(surv, long, Surv(L, R, type = "interval2") ~ x,
                      event_time = "Tev", long_sub_fixed = y ~ year + Tev + x,
                      long_sub_random = ~ year | id, time_variable = "year",
                      n_burnin = 2, n_imputations = 2, n_thin = 1, n_grid = 15,
                      seed = 1)
fit$pooled

# the fit plugs into the prediction functions; Tev is unknown for a new patient
history <- long[long$id == 1 & long$year <= 2, ]
history$Tev <- NA_real_
predictRisk(history, fit$long_fit_all, fit$survival_fit_all,
            prediction_time = 2, horizon = 2, time_variable = "year",
            survival_variable_all = NULL, survival_trans_function = NULL,
            bandcount1 = 10, bandcount2 = 20)
```

`imputeLongitudinal` *Impute missing longitudinal biomarker values*

Description

Fills in missing biomarker values before `longitudinalSub()` is fit, assuming they are missing at random (MAR): whether a value is missing depends only on what is observed.

It is often not needed. `longitudinalSub()` fits each biomarker's linear mixed model by maximum likelihood on all of that biomarker's observed values, which is already valid under MAR; only the joint covariance of the random effects across biomarkers, `Sigma_fit`, is estimated from the subjects with some value of every biomarker. In simulations on `pbc3` (see the "Missing data imputation" article on the package website), no imputation method made the predictions closer to those from the complete data than this complete-case analysis when visits were missing values at random; with half of the subjects never measured for a biomarker, multiple imputation with `method = "mixed"` was slightly closer. Imputation can also help when the missingness depends on variables outside the model, if they are added to the formulas used here.

Three methods are available, selected with `method`:

- `"mixed"` (default) draws each subject's missing values from the multivariate linear mixed model `longitudinalSub()` fits, given everything observed for that subject – all visits and all biomarkers – with the model's parameters drawn from an approximation to their posterior for each completion (proper multiple imputation; see `mixed_model_imputations()`). It needs no extra package, keeps the variation between subjects, and was the only method in the simulations that left the random-effects variances and the event-time coefficients unbiased.
- `"miwae"` fits a MIWAE (Missing data Importance-Weighted AutoEncoder; Mattei & Frellsen, 2019) – a variational autoencoder whose training objective only scores reconstruction on the observed entries – jointly across all supplied biomarkers, and uses self-normalized importance resampling from the fitted decoder to draw one or more plausible completions of the missing cells.

- `"diffusion"` fits a conditional denoising diffusion probabilistic model (DDPM; Ho, Jain & Abbeel, 2020) over the same per-row biomarker vector, trained as in CSDI (Tashiro et al., 2021): in every mini-batch, each row's observed entries are split at random into targets and conditions, and the network learns to denoise the targets given the conditions' values, the covariates and the observation mask, so the loss only ever involves observed entries. Missing cells are then filled by running the reverse diffusion with the missing entries as targets and all observed ones as conditions; observed entries are returned unchanged.

The two deep generative methods need the **torch** package. They complete each visit from that visit's covariates and observed biomarkers (and, with `history = TRUE`, a summary of the subject's other visits), and in the simulations they shrank the estimated variation between subjects, which every prediction uses; with `history = TRUE` they also biased the event-time coefficients when subjects lacked a biomarker entirely. The completed data.frame(s) of any method can be passed directly to `longitudinalSub()` in place of the original `data_fit_all`.

Usage

```
imputeLongitudinal(
  data_fit_all,
  long_sub_fixed,
  long_sub_random,
  time_variable,
  n_imputations = 5,
  latent_dim = 8,
  hidden_units = c(64, 32),
  epochs = 300,
  importance_samples = 20,
  diffusion_steps = 100,
  method = c("mixed", "miwae", "diffusion"),
  impute = c("single", "multiple"),
  history = FALSE,
  seed = NULL
)
```

Arguments

`data_fit_all`	As in `longitudinalSub`: either a single data.frame reused for every biomarker, or a list with one data.frame per biomarker (long format, one row per subject-visit).
`long_sub_fixed`	As in `longitudinalSub`: a formula, or list of formulas (one per biomarker), whose left-hand side must be a bare column name (no transformation) – the column that gets imputed.
`long_sub_random`	As in `longitudinalSub`: used only to read off the subject id variable (`~ ... | id`).
`time_variable`	Name of the visit-time column used, together with the id variable, to align rows across biomarkers.
`n_imputations`	Number of stochastic completions to draw. Also controls how many completed datasets are returned when `impute = "multiple"`.

`latent_dim` Dimension of the VAE's latent space. Only used when `method = "miwae"`.

`hidden_units` Integer vector giving the hidden-layer sizes of the network: the encoder (and, reversed, the decoder) when `method = "miwae"`, or the denoising network's body when `method = "diffusion"`.

`epochs` Number of training epochs: full-batch gradient steps for `method = "miwae"`, passes over the data in mini-batches of 128 rows for `method = "diffusion"`. The default suits data sets of a few hundred to a few thousand rows; many more epochs can overfit, which shows as imputations that vary too little between draws (and so pooled standard errors that are too small) – in a simulation, MIWAE's spread between draws fell from about the true residual SD at 300 epochs to 80\ of it at 3000.

`importance_samples` Number of importance samples (`K`) used both in the training objective and at imputation time. Only used when `method = "miwae"`.

`diffusion_steps` Number of forward/reverse diffusion timesteps. Only used when `method = "diffusion"`; more steps generally improve sample quality at proportionally higher training/sampling cost.

`method` `"mixed"` (default), `"miwae"` or `"diffusion"`; see Description. `latent_dim`, `hidden_units`, `epochs`, `importance_samples` and `diffusion_steps` only apply to the deep generative methods.

`impute` Either `"single"` (default; returns one completed `data_fit_all`, filling each missing cell with the mean of the `n_imputations` draws – a drop-in replacement for the original `data_fit_all`) or `"multiple"` (also returns `n_imputations` separately-drawn completed datasets in `data_fit_all_list`, to fit `longitudinalSub()` once per completion and pool the fits with `poolLongitudinalSub`). Prefer `"multiple"` unless only a little data is missing. The mean of several draws is less variable than the values it stands in for, so a model fit to the `"single"` completion treats the imputed cells as exactly known: its standard errors are too small, and its residual and random-effects variances – including `Sigma_fit`, which `predictRisk` and `predictLongitudinal` use directly – tend to be underestimated, more so the larger the share of imputed cells.

`history` For the deep generative methods: whether each visit's inputs include a summary of the same subject's other visits (see `history_features()`). Without it an imputed value does not carry the subject's own level, which shrinks the estimated variation between subjects; with it, that shrinkage is smaller when visits miss values at random, but in simulations it biased the event-time coefficients when subjects lacked a biomarker entirely. Defaults to `FALSE`.

`seed` Optional integer seed for reproducibility. It seeds both **torch** and R's random number generator for the duration of the call; R's random number state from before the call is restored afterwards.

Details

Only the biomarker (response) columns named on the left-hand side of `long_sub_fixed` are imputed, and they must be numeric (continuous): an ordinal biomarker cannot be imputed with this function, and gives an error. Every covariate referenced on the right-hand side of `long_sub_fixed`/`long_sub_random`

is assumed fully observed (an error is raised if any covariate has missing values). The deep generative methods model `p(biomarkers | covariates, missingness mask)` row by row (one row per subject-visit; the covariates include the visit time `time_variable` when `long_sub_fixed` uses it), and use `long_sub_random` only for the id variable; `"mixed"` uses the full random-effects structure.

Rows are aligned across biomarkers by `(id, time_variable)`: when `data_fit_all` is a list of different data.frames with different visit schedules per biomarker (rather than one data.frame shared by every biomarker), the union of every `(id, time_variable)` pair seen in any of them is used, so a visit at which one biomarker was measured but another was not still gets a value imputed for the missing one.

Because the deep generative methods need enough data to fit reliably; the `diagnostics` element of the return value also reports a classical single regression imputation (each biomarker regressed on the covariates, fit on its observed rows) as a lightweight baseline, so the two can be compared on a given dataset before trusting the deep model's completions.

Value

A named list with elements:

data_fit_all A list with one completed data frame per biomarker (also when `data_fit_all` was a single data frame), with every missing biomarker cell filled. Each holds one row per (id, `time_variable`) pair seen for any biomarker, and only the columns its model uses: the id, `time_variable`, the biomarker and the covariates of its `long_sub_fixed` formula – other columns of the input are not carried over. Ready to pass straight to `longitudinalSub` with the same formulas.

data_fit_all_list When `impute = "multiple"`, a list of `n_imputations` completed versions of `data_fit_all`, each from its own draw; `NULL` when `impute = "single"`.

diagnostics A named list, one element per biomarker (its `deep_generative` entry describes the completions of whichever `method` was used), each with the number of missing cells and observed-vs-imputed mean/sd for both the deep-generative and classical-baseline completions.

References

Mattei, P.-A. and Frellsen, J. (2019). MIWAE: Deep Generative Modelling and Imputation of Incomplete Data Sets. *Proceedings of the 36th International Conference on Machine Learning*, PMLR 97:4413-4423.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising Diffusion Probabilistic Models. *Advances in Neural Information Processing Systems*, 33:6840-6851.

Tashiro, Y., Song, J., Song, Y., and Ermon, S. (2021). CSDI: Conditional Score-based Diffusion Models for Probabilistic Time Series Imputation. *Advances in Neural Information Processing Systems*, 34:24804-24816.

Examples

```
data(pbc3)
data_fit_all <- pbc3[pbc3$status3 == 1, ]

# pbc3 has no missing serBilir/albumin values; delete some at random
```

```
# (MCAR, a special case of MAR) to have something to impute
set.seed(1)
n <- nrow(data_fit_all)
data_fit_all$serBilir[sample.int(n, floor(0.1 * n))] <- NA
data_fit_all$albumin[sample.int(n, floor(0.1 * n))] <- NA

long_sub_fixed <- list(
  "long1" = serBilir ~ year + age + sex + years,
  "long2" = albumin ~ year + age + sex + years)
long_sub_random <- list("long1" = ~ year | id, "long2" = ~ year | id)

# multiple imputation from the mixed model, pooled with Rubin's rules
imputed <- imputeLongitudinal(data_fit_all, long_sub_fixed, long_sub_random,
                              time_variable = "year", impute = "multiple", seed = 1)
fits <- lapply(imputed$data_fit_all_list, longitudinalSub,
               long_sub_fixed = long_sub_fixed, long_sub_random = long_sub_random)
pooled <- poolLongitudinalSub(fits)


# a deep generative method instead (needs torch)
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  imputed_miwae <- imputeLongitudinal(data_fit_all, long_sub_fixed, long_sub_random,
                                      time_variable = "year", method = "miwae",
                                      epochs = 50, seed = 1)
}
```

`longitudinalSub` *Fit a multivariate longitudinal sub-model*

Description

Fits one linear mixed-effects model per longitudinal biomarker separately (via `lme`), keeping each biomarker's own fixed-effects and residual-variance estimates from that separate fit. The `M` separate fits are then combined into a single multivariate model by re-estimating the full random-effects variance-covariance matrix jointly across all `M` biomarkers, via an EM algorithm initialized from the block-diagonal covariance implied by the separate fits – so correlation between biomarkers' random effects is captured, even though the fixed effects and residual variances themselves are not re-estimated jointly and remain exactly what each biomarker's own `lme()` fit produced. This lets you assess the effect of predictor variables on several longitudinal outcomes at once while accounting for both population-level (fixed) effects and subject-level (random) variability, and for how the outcomes covary within a subject. The result is one of the two sub-models – together with `survivalSub` – that `predictRisk`/`dynamicPredictionBio` combine to produce dynamic risk/biomarker predictions.

Usage

```
longitudinalSub(
```

```
  data_fit_all,
  long_sub_fixed,
  long_sub_random,
  biomarker_type = NULL
)
```

Arguments

data_fit_all	This process requires a set of data.frame objects designated for model fitting, with each data.frame representing a separate longitudinal outcome. These data.frame objects must include the variables identified in long_sub_fixed and long_sub_random. The use of a list arrangement facilitates the inclusion of various longitudinal outcomes, which may adhere to different measurement protocols. When all longitudinal outcomes are measured at the same time points for every patient, a single data.frame object can be in a list. It is assumed that every data.frame is organized in a long format.
long_sub_fixed	This refers to a collection of formulas detailing the fixed effects portion for each longitudinal outcome. On the left side of each formula, the response variable is defined, while the right side outlines the fixed effect terms. Should only a single formula be provided - whether as a list with one item or as a standalone formula - it is inferred that a conventional univariate joint model is being constructed. Terms whose basis/contrasts depend on the data they are computed from – poly() in its default orthogonal mode, splines::ns()/ splines::bs(), and factor() – are fine in a continuous biomarker's long_sub_fixed formula: the basis and factor levels the model was fit with are reused when estimating Sigma_fit and at prediction time. In long_sub_random, or in an ordinal biomarker's long_sub_fixed, they trigger a warning, because there the design matrix is still rebuilt from the data at hand (a single patient's rows at prediction time), so the basis can silently disagree with the one used to fit the model. Prefer poly(..., raw = TRUE), I(x^2), log(), sqrt(), or other terms that do not depend on the surrounding data there.
long_sub_random	A list of one-sided formulas that define the model for the random effects of each longitudinal outcome. The number of items in this list should match the length of formLongFixed.
biomarker_type	Optional character vector, one entry per longitudinal outcome, each either "continuous" or "ordinal". When supplied, it always takes priority. When NULL (the default), each biomarker's type is auto-detected from its own response column in data_fit_all: a factor/ordered factor response is treated as "ordinal", anything else as "continuous". If every biomarker is continuous (the original use case), fitting proceeds exactly as before with no behavior change whatsoever. If at least one biomarker is ordinal, that biomarker is instead fit with ordinal::clmm() (a probit cumulative link mixed model, requiring the optional **ordinal** package) and folded into the shared random-effects covariance matrix via a Gaussian-copula extension of the EM algorithm – see longitudinalSubCopula() for implementation details.

Value

An object of class "longitudinalSub.BJM", a named list with elements:

lfit A list of fitted univariate linear mixed models, one per longitudinal outcome, each obtained via lme.

Sigma_fit The estimated variance-covariance matrix of the random effects in the multivariate linear mixed model.

long_sub_fixed The long_sub_fixed argument, as supplied.

long_sub_random The long_sub_random argument, as supplied.

xlevels A list, one element per longitudinal outcome, of the factor levels observed in the full training data for that outcome. Used internally at prediction time so that poly()/ splines::ns()/splines::bs()/factor() terms in long_sub_fixed reuse the basis/contrasts fit at training time instead of recomputing one from a small, patient-specific slice of data.

Examples

```
long_sub_fixed = list(
  "long1" = serBilir ~ year + age + sex +  (years) + (years) * year,
  "long2" = prothrombin ~ year + age + sex + (years) + (years) * year,
  "long3" = albumin ~ year + age + age * year + sex + (years) + (years) * year,
  "long4" = alkaline ~ year + age + sex + (years) + (years) * year,
  "long5" = SGOT ~ year + age + sex + (years) + (years) * year,
  "long6" = platelets ~ year + age + sex + (years)  + (years) * year)

long_sub_random =list(
  "long1" =  ~ year| id,
  "long2" =  ~ year| id,
  "long3" =  ~ year| id,
  "long4" =  ~ year| id,
  "long5" =  ~ year| id,
  "long6" =  ~ year| id)

# Complete case analysis
data_fit_all = list()
for(i in seq_len(length(long_sub_fixed))){
  data_fit_all[[i]] = pbc3[pbc3$status3 == 1, ]
}

# fitting longitudinal submodel
long_fit_all = longitudinalSub(data_fit_all, long_sub_fixed, long_sub_random)

# poly() in its default orthogonal mode, splines::ns()/bs(), and
# factor() are safe in a continuous biomarker's long_sub_fixed: the
# terms/xlevels/contrasts fit on the full training data are cached and
# reused when estimating Sigma_fit and at prediction time, instead of
# being recomputed from each patient's small per-prediction slice.
long_fit_poly = longitudinalSub(
  pbc3[pbc3$status3 == 1, ],
  serBilir ~ year + poly(age, 2) + factor(sex) + years,
```

```
  ~ year | id)

# A few more nonlinear-term styles. None of these would trigger the
# warning even in long_sub_random, because their basis doesn't depend on the surrounding
# data at all -- there's simply nothing that could disagree between the
# full training data and the small per-patient slice used at prediction
# time.
data_fit_extra_terms = pbc3[pbc3$status3 == 1, ]
# A transformed-time column can also be built once with survivalTrans()
# and merged onto the fitting data ahead of time -- by the time
# long_sub_fixed sees it, it is just an ordinary numeric column, same as
# any of the other terms below.
data_fit_extra_terms$Tyears1 = survivalTrans(c(1, 3, 5, 7))$survival_trans_function[[1]](
  data_fit_extra_terms$year)

long_fit_nonlinear = list(
  longitudinalSub(data_fit_extra_terms, serBilir ~ year + I(year^2) + age + sex,
                  ~ year | id),
  longitudinalSub(data_fit_extra_terms, serBilir ~ year + I(year^2) * age + sex,
                  ~ year | id),
  longitudinalSub(data_fit_extra_terms, serBilir ~ year + log(year + 1) + age + sex,
                  ~ year | id),
  longitudinalSub(data_fit_extra_terms, serBilir ~ year + sqrt(year) + age + sex,
                  ~ year | id),
  longitudinalSub(data_fit_extra_terms, serBilir ~ poly(year, 2, raw = TRUE) + age + sex,
                  ~ year | id),
  longitudinalSub(data_fit_extra_terms, serBilir ~ year + Tyears1 + age + sex,
                  ~ year | id)
)
```

---

pbc3 *Mayo Clinic primary biliary cirrhosis data used as example code*

Description

The dataset originates from the Mayo Clinic trial on primary biliary cirrhosis (PBC) of the liver, carried out from 1974 to 1984. It includes data from 424 PBC patients who were referred to the Mayo Clinic within this decade and met the eligibility requirements for a randomized placebo-controlled trial of D-penicillamine. However, only the initial 312 cases from the dataset were enrolled in the randomized trial. Thus, the dataset specifically pertains to these 312 patients, for whom the data is largely complete.

Usage

```
data(pbc3)
```

Format

A data frame with 1945 observations on the following 27 variables:

`id` patients identifier; in total there are 312 patients.

`years` number of years between registration and the earlier of death, transplantation, or study analysis time.

`status` a factor with levels `alive`, `transplanted` and `dead`.

`drug` a factor with levels `placebo` and `D-penicil`.

`age` at registration in years.

`sex` a factor with levels `male` and `female`.

`year` number of years between enrollment and this visit date, remaining values on the line of data refer to this visit.

`ascites` a factor with levels `No` and `Yes`.

`hepatomegaly` a factor with levels `No` and `Yes`.

`spiders` a factor with levels `No` and `Yes`.

`edema` a factor with levels `No edema` (i.e. no edema and no diuretic therapy for edema), `edema no diuretics` (i.e. edema present without diuretics, or edema resolved by diuretics), and `edema despite diuretics` (i.e. edema despite diuretic therapy).

`serBilir` serum bilirubin in mg/dl.

`serChol` serum cholesterol in mg/dl.

`albumin` albumin in mg/dl.

`alkaline` alkaline phosphatase in U/liter.

`SGOT` SGOT in U/ml.

`platelets` platelets per cubic ml/1000.

`prothrombin` prothrombin time in seconds.

`histologic` histologic stage of disease.

`status2` a numeric vector with the value 1 denoting if the patient was dead, and 0 if the patient was alive or transplanted.

`status3` a numeric vector with the value 1 denoting if the patient was dead or transplanted, and 0 if the patient was alive.

`status4` a numeric vector with the value 1 denoting if the patient was transplanted, and 0 if the patient was dead. Used for competing risks.

`status5` a numeric vector with the value 2 if the patient was transplanted, 1 denoting if the patient was dead, and 0 if the patient was alive. Used for competing risks with censored.

`Tyears1` a numeric vector with a transformed value of time-to-event outcome.

`Tyears2` a numeric vector with a transformed value of time-to-event outcome.

`Tyears3` a numeric vector with a transformed value of time-to-event outcome.

`Tyears4` a numeric vector with a transformed value of time-to-event outcome.

Source

pbc.

References

Fleming T, Harrington D. *Counting Processes and Survival Analysis*. 1991; New York: Wiley.

Therneau T, Grambsch P. *Modeling Survival Data: Extending the Cox Model*. 2000; New York: Springer-Verlag.

`performancePlot` *Plot predictive performance across landmark times*

Description

Evaluates how well the backward joint model's dynamic risk predictions discriminate and are calibrated, at each of several landmark times. For each landmark `s` in `prediction_time`, every subject in `data_predict_all` still event-free at `s` gets a predicted risk from `predictRisk`, using only their longitudinal history up to `s`, of an event in `(s, s + horizon]`; this is compared with what actually happened, by

AUC the time-dependent (cumulative/dynamic) area under the ROC curve: the probability that a subject with an event in the window has a higher predicted risk than one still event-free at `s + horizon`. 0.5 is no better than chance, 1 is perfect.

Brier score the mean squared difference between the predicted risk and the observed event indicator in the window. Lower is better.

Subjects censored within the window are handled by inverse probability of censoring weighting (IPCW), with the censoring distribution among the subjects at risk at `s` estimated by Kaplan–Meier. With competing risks, both measures are computed separately for each event type: the cases are the subjects with that event type in the window, and for the Brier score a subject with the other event type in the window counts as not having the event.

Evaluating on the data the sub-models were fit on gives an optimistic (apparent) performance; pass held-out validation data in `data_predict_all` when available.

Usage

```
performancePlot(
  data_predict_all,
  long_fit_all,
  survival_fit_all,
  prediction_time,
  horizon,
  time_variable,
  survival_variable_all,
  survival_trans_function,
  bandcount1 = "auto",
  bandcount2 = "auto",
  interval_method = c("model", "ipcw")
)
```

Arguments

`data_predict_all`
: The evaluation data, in the same format as for `predictRisk`: a list of long-format `data.frames`, one per longitudinal outcome (or a single `data.frame` used for all). Unlike for `predictRisk()`, it must also contain each subject's observed outcome: the survival time and status variables of `survival_fit_all`'s Cox formula and, with competing risks, the event type variable of its `form_conditional_cr`. The outcome is taken from the first data frame and is hidden from `predictRisk()`.

`long_fit_all`
: A `longitudinalSub.BJM` object from `longitudinalSub`.

`survival_fit_all`
: A `survivalSub.BJM` object from `survivalSub`.

`prediction_time`
: A vector of landmark times.

`horizon`
: The prediction horizon (a single positive number): the window after each landmark in which events are counted.

`time_variable`
: The name of the time variable in the linear mixed models.

`survival_variable_all`, `survival_trans_function`
: As for `predictRisk`.

`bandcount1`, `bandcount2`
: As for `predictRisk`; passed to it at each landmark. `"auto"` (the default) retunes them in every call, which over a whole cohort is several times slower than fixed values (about 10 times for 160 patients in `pbc3`): for a large evaluation, choose them once on a few patients with `checkBandcountConvergence` and pass the numbers.

`interval_method`
: Only for an interval-censored `survival_fit_all`: `"model"` (default) or `"ipcw"`; see Description. For an interval-censored fit, `data_predict_all` must contain the interval columns of `form_marginal_surv` (and the event type) instead of a survival time and status.

Value

A `ggplot` object with one panel per measure, the landmark time on the horizontal axis. Its `data` element is a `data.frame` with one row per landmark, measure (and event type), and columns `landmark`, `measure`, `cause`, `value`, `n_at_risk` (subjects predicted) and `n_cases` (events of that type observed in the window).

Interval-censored fits (experimental)

The event is then only known to lie between the last negative visit L and the visit R that detected it. A subject is at risk at `s` when followed up past `s` without a detected event, and the predicted risk is that of `predictRisk()` (conditional on being event-free at the last visit). Following Yang, Rizopoulos, Newcomb and Erler (2026, Biometrical Journal 68:e70108), `interval_method` chooses how subjects whose interval straddles `s` or `s + horizon` are handled:

- `"model"` (default): every subject at risk counts, as a case or a control with the probability the fitted model gives their event time given their observed interval. The model only splits

the interval observed for each subject, so in our simulations (visits every 1–3 years, 2-year window) both measures were within about 0.003 of their true values, also with a survival sub-model that left out a covariate. The model does grade itself, though: for a deliberately miscalibrated model the Brier score came out slightly too good (by about 0.003), so judge calibration with `calibrationPlot`, which does not use the model.

- `"ipcw"`: only subjects whose status is certain are used – cases detected inside the window after a negative visit at or after `s`, controls with a negative visit at or after `s + horizon` – weighted by the inverse Kaplan–Meier probability of still being followed up. It does not use the model, but these weights do not account for how likely the status of a subject is to be certain, which depends on the visit schedule: in the same simulations only about 5 of 60 cases were certain, the Brier score came out at about 0.03 instead of 0.13, and the AUC was about 0.01 too high with three times the error of `"model"`. With visits every 3–9 months the Brier score was still about 25% too low. Use it as a model-free check of the AUC.

Examples

```
data(pbc3)
survival_fit_all <- survivalSub(pbc3[!duplicated(pbc3$id), ],
                                Surv(years, status3) ~ age + sex, NULL)
long_sub_fixed <- list("long1" = serBilir ~ year + age + sex + years)
long_sub_random <- list("long1" = ~ year | id)
long_fit_all <- longitudinalSub(list(pbc3[pbc3$status3 == 1, ]),
                                long_sub_fixed, long_sub_random)
trans <- survivalTrans(c(1, 3, 5, 7))

# apparent performance on the fitting data, 2-year window
performancePlot(pbc3, long_fit_all, survival_fit_all,
  prediction_time = c(1, 3, 5), horizon = 2, time_variable = "year",
  trans$survival_variable_all, trans$survival_trans_function)
```

`plot.longitudinalSub.BJM`

Plot a fitted longitudinal sub-model

Description

Diagnostic plots for the per-biomarker mixed models and the shared random-effects covariance of a `longitudinalSub` fit, one panel per biomarker:

`"residuals"` Standardized (Pearson) residuals against fitted values, with a LOESS smoother; a trend or a funnel shape suggests a misspecified mean or non-constant variance.

`"qq"` Normal Q-Q plot of the standardized residuals.

`"ranef"` Normal Q-Q plot of each predicted random effect (the model assumes they are normal).

`"corr"` Heat map of the correlation matrix of the multivariate random-effects covariance `Sigma_fit`, i.e. how the subject-level deviations of the different biomarkers move together.

Residual plots are drawn for continuous biomarkers only; an ordinal biomarker (fit by `ordinal::clmm()`) has no residuals on the response scale and is left out of them with a message.

Usage

```
## S3 method for class 'longitudinalSub.BJM'
plot(x, which = c("residuals", "qq", "ranef", "corr"), ...)
```

Arguments

x	A longitudinalSub.BJM object returned by longitudinalSub.
which	Which plot to draw: "residuals" (default), "qq", "ranef" or "corr".
...	Currently unused.

Value

A ggplot object.

Examples

```
data(pbc3)
long_sub_fixed <- list("long1" = serBilir ~ year + age + sex + years,
                       "long2" = albumin ~ year + age + sex + years)
long_sub_random <- list("long1" = ~ year | id, "long2" = ~ year | id)
data_fit <- pbc3[pbc3$status3 == 1, ]
long_fit_all <- longitudinalSub(list(data_fit, data_fit), long_sub_fixed, long_sub_random)

plot(long_fit_all)
plot(long_fit_all, which = "qq")
plot(long_fit_all, which = "ranef")
plot(long_fit_all, which = "corr")
```

---

plot.predictLongitudinal.BJM

Plot predicted biomarker distributions

Description

Plots the predicted distribution of the biomarker at prediction_time + horizon returned by predictLongitudinal, one curve per patient, with a dashed line at the patient's point prediction (the most likely value, Y_predict). For an ordinal biomarker, plots each patient's predicted category probabilities as bars instead.

Usage

```
## S3 method for class 'predictLongitudinal.BJM'
plot(x, subject = NULL, ...)
```

Arguments

x	A predictLongitudinal.BJM object, i.e. the result of predictLongitudinal() for a single biomarker. For several biomarkers, plot one element of the returned list, e.g. plot(result[["serBilir"]]).
subject	Patients to plot, as ids (the names of x$Y_predict) or positions. NULL (default) plots every patient, or the first six when there are more.
...	Currently unused.

Value

A ggplot object.

Examples

```
data(pbc3)
survival_fit_all <- survivalSub(pbc3[!duplicated(pbc3$id), ],
                                Surv(years, status3) ~ age + sex, NULL)
long_sub_fixed <- list("long1" = serBilir ~ year + age + sex + years)
long_sub_random <- list("long1" = ~ year | id)
long_fit_all <- longitudinalSub(list(pbc3[pbc3$status3 == 1, ]),
                                long_sub_fixed, long_sub_random)

data_predict <- pbc3[pbc3$id %in% c(2, 4, 5) & pbc3$year <= 3, ]
data_predict$years <- NA
trans <- survivalTrans(c(1, 3, 5, 7))
pred <- predictLongitudinal(data_predict, long_fit_all, survival_fit_all,
  prediction_time = 3, horizon = 1, time_variable = "year",
  trans$survival_variable_all, trans$survival_trans_function, bio_i = 1)
plot(pred)
```

---

plot.simulateTrajectory.BJM

Plot simulated futures

Description

Plots the draws of simulateTrajectory, pooling every draw of the patients chosen by id (all of them by default: one patient's draws picture that patient's predictive distribution, one draw each of a synthetic cohort pictures the cohort):

"trajectory" One panel per biomarker. For a continuous biomarker, n_paths of the simulated trajectories as thin lines, with the median and the central level interval of the draws at each time, and the measurements conditioned on as points. For an ordinal biomarker, the share of the draws in each category at each time. With truncate = TRUE in simulateTrajectory() the summaries at a time describe the draws still event-free then.

"event" The cumulative incidence of the event (of each event type, with competing risks) after prediction_time, computed from the drawn event times.

Usage

```
## S3 method for class 'simulateTrajectory.BJM'
plot(
  x,
  which = c("trajectory", "event"),
  id = NULL,
  n_paths = 30,
  level = 0.9,
  ...
)
```

Arguments

x	A simulateTrajectory.BJM object returned by simulateTrajectory.
which	"trajectory" (default) or "event".
id	Patient id(s) whose draws to plot; NULL (default) for all.
n_paths	Number of simulated trajectories drawn as lines (the first ones; the draws are independent, so they are a random sample).
level	Coverage of the interval drawn around the median.
...	Currently unused.

Value

A ggplot object.

Examples

```
data(pbc3)
survival_fit_all <- survivalSub(pbc3[!duplicated(pbc3$id), ],
                                Surv(years, status3) ~ age + sex, NULL)
long_fit_all <- longitudinalSub(list(pbc3[pbc3$status3 == 1, ], pbc3[pbc3$status3 == 1, ]),
                                list(serBilir ~ year + age + sex + years,
                                     albumin ~ year + age + sex + years),
                                list(~ year | id, ~ year | id))
history <- pbc3[pbc3$id == 2 & pbc3$year <= 3, ]
sims <- simulateTrajectory(history, long_fit_all, survival_fit_all,
                           prediction_time = 3, times = seq(3, 8, by = 0.5),
                           time_variable = "year", survival_variable_all = list(),
                           survival_trans_function = list(), n_sim = 200, seed = 1)
plot(sims)
plot(sims, which = "event")
```

`plot.survivalSub.BJM` *Plot a fitted survival sub-model*

Description

`"forest"` Forest plot of the hazard ratios (with 95\ confidence intervals) of the Cox marginal survival model and, when the fit has competing risks, of the odds ratios (with 95\ intervals) of the logistic event-type model, on a log scale with a reference line at 1.

`"basehaz"` The Cox model's baseline cumulative hazard (covariates at 0, not centered), one step curve per stratum.

Usage

```
## S3 method for class 'survivalSub.BJM'
plot(x, which = c("forest", "basehaz"), ...)
```

Arguments

`x`	A `survivalSub.BJM` object returned by `survivalSub`.
`which`	Which plot to draw: `"forest"` (default) or `"basehaz"`.
`...`	Currently unused.

Value

A `ggplot` object.

Examples

```
data(pbc3)
data_survival_fitting <- pbc3[!duplicated(pbc3$id), ]
survival_fit_all <- survivalSub(data_survival_fitting,
                                Surv(years, status3) ~ age + sex,
                                status4 ~ years + age + sex)
plot(survival_fit_all)
plot(survival_fit_all, which = "basehaz")
```

poolLongitudinalSub *Pool longitudinal sub-model fits across multiple imputations with Rubin's rules*

Description

When imputeLongitudinal is run with impute = "multiple", it returns n_imputations independently completed versions of data_fit_all in data_fit_all_list, instead of a single completed dataset filled with the across-draw mean (impute = "single"). Fitting longitudinalSub once per completion and combining the resulting fixed-effect estimates with Rubin's rules (Rubin, 1987) – rather than averaging into one completed dataset up front – is the standard way multiple imputation propagates the extra uncertainty from not knowing the true missing values into the final standard errors; impute = "single" does not do this (see the "Missing data imputation" article on the package website). poolLongitudinalSub() performs that combination step.

Usage

```
poolLongitudinalSub(long_fit_all_list)
```

Arguments

long_fit_all_list
: A list of two or more longitudinalSub.BJM objects: the result of calling longitudinalSub once on each completed dataset in imputeLongitudinal(..., impute = "multiple")$data_fit_all_list, using the same long_sub_fixed/long_sub_random for every completion.

Details

For each fixed-effect coefficient, across the m fits in long_fit_all_list:

- the pooled estimate is the mean of the m per-completion estimates ($\bar{Q}$);
- the pooled variance (T) adds the average within-imputation variance ($\bar{U}$, from each fit's own standard error) to the between-imputation variance (B, the sample variance of the m estimates) inflated by a factor of $1 + 1/m$: $T = \bar{U} + (1 + 1/m)B$;
- the pooled degrees of freedom use the Barnard & Rubin (1999) adjustment, which interpolates between the classical Rubin (1987) degrees of freedom (based only on m and the fraction of missing information) and the complete-data degrees of freedom reported by each nlme::lme() fit. This adjustment matters most at the small sample sizes typical of this package's use case, where the classical, unadjusted degrees of freedom can otherwise come out larger than the complete-data degrees of freedom, which is not sensible.

A large fraction of missing information (fmi, close to 1) for a coefficient means the missing data cost a lot of precision for that coefficient specifically, even if the biomarker's overall missingness rate is modest.

Value

An object of class "poolLongitudinalSub.BJM", a named list with elements:

pooled A list, one data.frame per longitudinal outcome, with one row per fixed-effect coefficient and columns estimate, std_error, df, statistic, p_value, riv (relative increase in variance due to missingness), and fmi (fraction of missing information).

m The number of completed-data fits pooled.

long_sub_fixed The long_sub_fixed used by every fit (taken from the first element of long_fit_all_list).

long_fit_all A longitudinalSub.BJM object usable for prediction (e.g. as long_fit_all in predictRisk): the first fit, with each biomarker's fixed effects replaced by their pooled estimates, its residual variance by the average across completions, and Sigma_fit by the average of the m fits' Sigma_fit (Rubin's rules point estimates). Averaging the m fits' predictions instead is a common alternative.

References

Rubin, D. B. (1987). *Multiple Imputation for Nonresponse in Surveys*. John Wiley & Sons.

Barnard, J. and Rubin, D. B. (1999). Small-Sample Degrees of Freedom with Multiple Imputation. *Biometrika*, 86(4):948-955.

Examples

```
if (requireNamespace("torch", quietly = TRUE) && torch::torch_is_installed()) {
  data(pbc3)
  data_fit_all <- pbc3[pbc3$status3 == 1, ]

  set.seed(1)
  n <- nrow(data_fit_all)
  data_fit_all$serBilir[sample.int(n, floor(0.1 * n))] <- NA
  data_fit_all$albumin[sample.int(n, floor(0.1 * n))] <- NA

  long_sub_fixed <- list(
    "long1" = serBilir ~ year + age + sex + years,
    "long2" = albumin ~ year + age + sex + years)
  long_sub_random <- list("long1" = ~ year | id, "long2" = ~ year | id)

  imputed <- imputeLongitudinal(data_fit_all, long_sub_fixed, long_sub_random,
                                time_variable = "year", n_imputations = 5,
                                impute = "multiple", epochs = 50, seed = 1)

  long_fit_all_list <- lapply(imputed$data_fit_all_list, function(d) {
    longitudinalSub(d, long_sub_fixed, long_sub_random)
  })
  pooled <- poolLongitudinalSub(long_fit_all_list)
  pooled
}
```

predictLongitudinal *Dynamic prediction function for future longitudinal outcomes*

Description

Companion to predictRisk, using the same fitted longitudinal (longitudinalSub) and survival (survivalSub) sub-models, but instead of an event-risk probability this returns a predictive density for a future value of one or more chosen biomarkers (bio_i) at prediction_time + horizon, conditional on the subject's observed longitudinal history up to prediction_time and on survival (being event-free) through prediction_time + horizon, the time the biomarker is predicted at.

This is the single, unified entry point for biomarker-value prediction:

- If bio_i names **exactly one** biomarker, this predicts just that biomarker and returns a single "predictLongitudinal.BJM" object.
- If bio_i names **more than one** biomarker, or is left at its default NULL (meaning every biomarker in long_fit_all), this instead predicts every requested biomarker and returns a named list of "predictLongitudinal.BJM" objects (classed "predictLongitudinalAll.BJM"), computing the bio_i-**independent** part of the pipeline (restricting to at-risk patients, the survival-side integration grid, and the denominator conditional density) only **once** and reusing it across every biomarker, instead of recomputing it once per biomarker. This matters most under the Gaussian-copula path (see longitudinalSubCopula), where that denominator involves mvtnorm::pmvnorm() Monte-Carlo evaluations that are otherwise the dominant cost of a multi-biomarker prediction run.

See Value for the exact return shape in each case.

bio_i may refer to either **continuous** or **ordinal** biomarkers (see longitudinalSubCopula), and a single call may mix both. For an ordinal biomarker, Y_all/Y_predict hold integer *category codes* (1:K, in the fitted factor's levels()/threshold order) rather than a numeric grid, with Y_all additionally carrying a "category_labels" attribute giving the matching level-label strings; bandcount3 is ignored for that biomarker, since its candidate grid is fixed at its category count (see Details).

The prediction is conditional on the longitudinal history observed up to prediction_time: rows of data_predict_all whose time_variable is later than prediction_time are dropped, with a warning, before predicting.

bandcount2 (controlling the shared survival-integration grid) and bandcount3 (controlling each biomarker's own candidate-value grid) are auto-tuned separately when left at their default "auto", via the doubling-until-stable check described in Details. When bio_i names more than one biomarker, bandcount2 is resolved **once** using a single representative biomarker (the first one in bio_i), and bandcount3 is then resolved independently for **every** requested biomarker (their candidate-value grids need not converge at the same resolution) – except for any **ordinal** biomarker, for which bandcount3 tuning is always skipped (see above) and "bandcount3" is recorded as NA for that biomarker.

Usage

```
predictLongitudinal(
  bio_i = NULL,
```

```
  data_predict_all,
  long_fit_all,
  survival_fit_all,
  prediction_time,
  horizon,
  time_variable,
  survival_variable_all,
  survival_trans_function,
  bandcount2 = "auto",
  bandcount3 = "auto",
  n_cores = 1
)
```

Arguments

bio_i	Integer vector of biomarkers to predict. May include continuous and/or ordinal biomarkers (see Details). Defaults to NULL, meaning every biomarker in long_fit_all. Naming exactly one biomarker returns a single result rather than a list of one – see Value.
data_predict_all	This involves a collection of data.frame objects for dynamic prediction, each corresponding to a distinct longitudinal outcome. These data frames should contain the variables specified in long_sub_fixed and long_sub_random. Utilizing a list structure allows for the incorporation of multiple longitudinal outcomes, each potentially following different measurement protocols. In instances where all longitudinal outcomes are recorded at identical time points across patients, a singular data.frame object may be used in a list. Alternatively, a single bare data.frame (not wrapped in a list) may be supplied directly; it is then reused for every longitudinal outcome. It is presumed that each data frame is structured in a long format.
long_fit_all	Outputs from the model fitting process using the nlme package, encompassing the results and parameters obtained from the analysis.
survival_fit_all	Results and parameters generated from the model fitting procedure, utilizing the coxph function. These outputs include the comprehensive findings and variables derived from the analysis.
prediction_time	Time used to make the prediction
horizon	Prediction horizon
time_variable	The name of time variable in linear mixed model.
survival_variable_all	The name of the transformed time-to-event outcomes variable.
survival_trans_function	The transformation function used for time-to-event outcomes, in the order of survival_variable_all.
bandcount2	The number of grid points spanning [prediction_time + horizon, upper_bound], where upper_bound is set internally as the earliest time by which every at-risk

patient's model-based probability of still being event-free (given event-free at `prediction_time + horizon`) has dropped below `1e-4`; this approximates integrating out to infinity for the denominator that normalizes the predicted density. A wider follow-up range needs a larger `bandcount2` to keep the grid spacing comparable. Defaults to `"auto"` (see Details).

`bandcount3` The number of points in the candidate-biomarker-value grid (`Y_all`) used to build the predicted density (`Y_density`) and locate its mode (`Y_predict`). This controls the resolution of the density curve, not a time integral; increase it if the density looks jagged or `Y_predict` jumps erratically between nearby grid points. Defaults to `"auto"` (see Details). Ignored for an ordinal biomarker (see Details) – that candidate grid is always the biomarker's fixed category count.

`n_cores` Number of CPU cores. With an ordinal biomarker the prediction's cost is per patient, and the patients are spread over `n_cores` cores with `parallel::mclapply()`; this is only parallel on Unix-like systems (Linux, macOS), and runs serially on Windows. The result is exactly the same for any `n_cores`. With continuous biomarkers only, predictions are fast and `n_cores` has no effect. Defaults to `1`.

Details

There is no universal correct value for `bandcount2`/`bandcount3`: as a practical check, double both and confirm the results barely change; if they do, keep doubling. By default (`bandcount2 = "auto"`, `bandcount3 = "auto"`), this doubling check is done for you: starting from small built-in values, both are doubled together, and the result is compared to the previous round, until the largest relative change in `Y_predict` drops below 1%, or 2 doublings have been tried (so at most 3 calls' worth of work). If it still has not converged by then, a warning reports this and the result at the largest value tried is returned anyway (not an error), so this never silently loops for an unbounded amount of time. Pass an explicit number for either argument to skip auto-tuning it and use a fixed value instead, or call `checkBandcountConvergence()` directly for more control over the tolerance and doubling count. See also `vignette("BJM-intro", package = "BJM")` for a worked example.

The integration grid, and the candidate-value grid of a continuous biomarker (which spans the observed values of every patient predicted), are shared by all patients predicted in one call, so a patient's value can differ slightly, within the grids' discretization error, depending on which other patients are predicted alongside it.

The denominator integrates over every event time after `prediction_time`, including times beyond the last follow-up time in the data `survivalSub()` was fit on. There, the baseline cumulative hazard is extrapolated linearly (a constant hazard), and the longitudinal sub-model's mean is evaluated at event times it was never fit on. When much of an at-risk patient's survival probability lies beyond the last follow-up time (e.g. a prediction late in follow-up), the prediction depends on this extrapolation; in `pbc3` it moved the risks checked by less than one percentage point.

For a fit with an ordinal biomarker, a patient's ordinal measurements contribute the probability of their observed categories. With a single ordinal random effect (e.g. one ordinal biomarker with a random intercept) it is computed by Gauss-Hermite quadrature, and is exact; with more, by Monte Carlo (`mvtnorm::pmvnorm()`), with an error around the fourth significant digit. Each patient's Monte Carlo draws use a fixed seed derived from the patient's id, so the results are the same on every run and for any `n_cores`, without `set.seed()`, and the caller's random-number state is left unchanged.

Value

If `bio_i` names exactly one biomarker: an object of class `"predictLongitudinal.BJM"`, a named list with elements:

Y_predict A vector, one entry per at-risk patient (named by patient id), giving the MAP (most likely) predicted value of the biomarker at `prediction_time + horizon`. For an ordinal biomarker, this is an integer category code (see Details), not a raw value.

Y_density A probability matrix whose rows correspond to the candidate biomarker values in `Y_all` and whose columns correspond to individual patients; each entry is the dynamically predicted density of the biomarker taking that value. For an ordinal biomarker, each row is instead that category's predicted probability.

Y_all The grid of candidate biomarker values used to build `Y_density`. For an ordinal biomarker, this is the vector of integer category codes `1:K`, carrying a `"category_labels"` attribute with the matching level-label strings (see Details).

Otherwise (`bio_i` names more than one biomarker, or is `NULL`): a named list of such objects, one per requested biomarker, named by that biomarker's response-variable name; with attributes `"bandcount2"` (the single resolved/used `bandcount2`) and `"bandcount3"` (a named numeric vector of the resolved/used `bandcount3` for each biomarker, or `NA` for an ordinal biomarker). Classed `"predictLongitudinalAll.BJM"`.

Examples

```
data(pbc3)

data_survival_fitting =  pbc3[!duplicated(pbc3$id), ]

form_marginal_surv = Surv(years, status3) ~ age + sex
form_conditional_cr = NULL

survival_fit_all = survivalSub(data_survival_fitting, form_marginal_surv,
                               form_conditional_cr)

long_sub_fixed = list(
  "long1" = serBilir ~ year + age + sex +  (years) + (years) * year,
  "long2" = prothrombin ~ year + age + sex + (years) + (years) * year,
  "long3" = albumin ~ year + age + age * year + sex + (years) + (years) * year,
  "long4" = alkaline ~ year + age + sex + (years) + (years) * year,
  "long5" = SGOT ~ year + age + sex + (years) + (years) * year,
  "long6" = platelets ~ year + age + sex + (years)  + (years) * year)

long_sub_random =list(
  "long1" =  ~ year| id,
  "long2" =  ~ year| id,
  "long3" =  ~ year| id,
  "long4" =  ~ year| id,
  "long5" =  ~ year| id,
  "long6" =  ~ year| id)

survival_variable_all = list(
```

```
  "Tyears1",  "Tyears2", "Tyears3", "Tyears4"
)

survival_trans_function = list(
  fun1 = function(x){abs(x - 1)},
  fun2 = function(x){abs(x - 3)},
  fun3 = function(x){abs(x - 5)},
  fun4 = function(x){abs(x - 7)}
)

# Complete case analysis
data_fit_all = list()
for(i in seq_len(length(long_sub_fixed))){
  data_fit_all[[i]] = pbc3[pbc3$status3 == 1, ]
}

# fitting longitudinal submodel
long_fit_all = longitudinalSub(data_fit_all, long_sub_fixed, long_sub_random)

i_PID = 2
data.raw.predict.1 = pbc3[pbc3$id == i_PID, ]

data_predict_all = list()
for(i in seq_len(length(long_sub_fixed))){
  data_predict_all[[i]] = data.raw.predict.1[data.raw.predict.1$year <= 3,]
}

# A single biomarker -> a single predictLongitudinal.BJM result
Y_predict = predictLongitudinal(bio_i = 1, data_predict_all, long_fit_all,
                                survival_fit_all, prediction_time = 3,
                                horizon = 3, time_variable = "year",
                                survival_variable_all, survival_trans_function,
                                bandcount2 = 40, bandcount3 = 400)

# Multiple biomarkers (or bio_i = NULL for all of them) -> a named list
Y_predict_all = predictLongitudinal(bio_i = NULL, data_predict_all, long_fit_all,
                                    survival_fit_all, prediction_time = 3,
                                    horizon = 3, time_variable = "year",
                                    survival_variable_all, survival_trans_function,
                                    bandcount2 = 40, bandcount3 = 400)
```

`predictPlot` *Plot risk and future biomarker predictions across a horizon sweep*

Description

Fixes the landmark time at `prediction_time` and sweeps forward over a range of prediction horizons, calling `predictRisk` (and, when `bio_pred` is supplied, `predictLongitudinal`) at each hori-

zon to trace out how predicted event risk (and, optionally, the predicted density/point-forecast of biomarker bio_pred) evolves the further out the prediction window extends from a single fixed landmark. This contrasts with riskPlot, which instead fixes the horizon and sweeps over a set of landmark times.

Usage

```
predictPlot(
  data_predict_all_one,
  long_fit_all,
  survival_fit_all,
  prediction_time = 4,
  horizon = seq(0, 3, 0.5),
  time_variable,
  survival_variable_all,
  survival_trans_function,
  bandcount1 = "auto",
  bandcount2 = "auto",
  bandcount3 = "auto",
  bio_his = 1,
  bio_pred = 1,
  density = 1,
  n_cores = 1
)
```

Arguments

data_predict_all_one	This involves a collection of data.frame one object for dynamic prediction and making plots, each corresponding to a distinct longitudinal outcome. These data frames should contain the variables specified in long_sub_fixed and long_sub_random. Utilizing a list structure allows for the incorporation of multiple longitudinal outcomes, each potentially following different measurement protocols. In instances where all longitudinal outcomes are recorded at identical time points across patients, a singular data.frame object may be used in a list. Alternatively, a single bare data.frame (not wrapped in a list) may be supplied directly; it is then reused for every longitudinal outcome. It is presumed that each data frame is structured in a long format.
long_fit_all	Outputs from the model fitting process using the nlme package, encompassing the results and parameters obtained from the analysis.
survival_fit_all	Results and parameters generated from the model fitting procedure, utilizing the coxph function. These outputs include the comprehensive findings and variables derived from the analysis.
prediction_time	Time used to make the prediction.
horizon	Prediction horizon.
time_variable	The name of time variable in linear mixed model.

survival_variable_all
: The name of the transformed time-to-event outcomes variable.

survival_trans_function
: The transformation function used for time-to-event outcomes, in the order of survival_variable_all.

bandcount1
: The number of grid points spanning the prediction window, from prediction_time to prediction_time + horizon. Larger values give a more accurate but slower estimate. Defaults to "auto", which resolves it once, before looping over horizon (using the largest requested horizon as a representative probe), by doubling from a built-in starting value until the predicted risk stabilizes; see predictRisk's bandcount1 for details of that search. The resolved value is then reused, fixed, for every point in horizon – it is not re-searched on every iteration.

bandcount2
: The number of grid points used to approximate integrating out to infinity when normalizing the predicted risk/density. A wider follow-up range needs a larger bandcount2 to keep the grid spacing comparable. Defaults to "auto"; resolved the same way as bandcount1 (jointly with it, when both are "auto").

bandcount3
: The number of points in the candidate-biomarker-value grid used to build the predicted density curve; controls the resolution of the density, not a time integral. Defaults to "auto"; resolved the same way, but only when bio_pred is non-NULL (it is unused otherwise).

  Pass explicit numbers instead of "auto" for full manual control, or use checkBandcountConvergence() (applied to predictRisk()/ predictLongitudinal() directly) to inspect the convergence behavior yourself. See also vignette("BJM-intro", package = "BJM") for further guidance on choosing bandcount1/bandcount2/ bandcount3.

bio_his
: Which biomarker history will be plotted

bio_pred
: Indicator, predict future biomarker or not, if NULL do not predict

density
: Indicator, plot future biomarker density or not, if NULL do not plot. For an ordinal biomarker the prediction is instead drawn as each category's predicted probability (points sized by it), with the axis labelled by category.

n_cores
: Number of CPU cores to use for computing the prediction at each point in horizon. Each point is computed independently, so this loop can be dispatched across cores. Defaults to 1 (serial execution; identical behavior/output to versions of this function without this argument). Values greater than 1 use parallel::mclapply(), which relies on forking and is therefore only actually parallel on Unix-like systems (Linux, macOS); on Windows, mclapply() silently runs the iterations serially regardless of n_cores (a limitation of R's fork-based parallelism, not of this package). Parallel execution produces exactly the same numeric result as serial execution – only the order in which iterations are computed (not the order results are assembled in) changes.

Value

Plot of risk and future biomarker with density using dynamic prediction.

Examples

```
data(pbc3)
```

```
data_survival_fitting =  pbc3[!duplicated(pbc3$id), ]

form_marginal_surv = Surv(years, status3) ~ age + sex
form_conditional_cr = NULL

survival_fit_all = survivalSub(data_survival_fitting, form_marginal_surv,
                               form_conditional_cr)

long_sub_fixed = list(
  "long1" = serBilir ~ year + age + sex +  (years) + (years) * year,
  "long2" = prothrombin ~ year + age + sex + (years) + (years) * year,
  "long3" = albumin ~ year + age + age * year + sex + (years) + (years) * year,
  "long4" = alkaline ~ year + age + sex + (years) + (years) * year,
  "long5" = SGOT ~ year + age + sex + (years) + (years) * year,
  "long6" = platelets ~ year + age + sex + (years)  + (years) * year)

long_sub_random =list(
  "long1" =  ~ year| id,
  "long2" =  ~ year| id,
  "long3" =  ~ year| id,
  "long4" =  ~ year| id,
  "long5" =  ~ year| id,
  "long6" =  ~ year| id)

survival_variable_all = list(
  "Tyears1",  "Tyears2", "Tyears3", "Tyears4"
)

survival_trans_function = list(
  fun1 = function(x){abs(x - 1)},
  fun2 = function(x){abs(x - 3)},
  fun3 = function(x){abs(x - 5)},
  fun4 = function(x){abs(x - 7)}
)

# Complete case analysis
data_fit_all = list()
for(i in seq_len(length(long_sub_fixed))){
  data_fit_all[[i]] = pbc3[pbc3$status3 == 1, ]
}

# fitting longitudinal submodel
long_fit_all = longitudinalSub(data_fit_all, long_sub_fixed, long_sub_random)

i_PID = 2
data.raw.predict.plot = pbc3[pbc3$id == i_PID, ]
data_predict_all_one = list(data.raw.predict.plot, data.raw.predict.plot, data.raw.predict.plot,
                    data.raw.predict.plot, data.raw.predict.plot, data.raw.predict.plot)

# plot biomarker 1 history,  predict future biomarker

predictPlot(data_predict_all_one, long_fit_all, survival_fit_all,
```

```
  prediction_time = 5, bio_his = 1, bio_pred = 1,
  horizon = seq(0.5, 3.0, 0.5), time_variable = "year",
  survival_variable_all, survival_trans_function,
bandcount1 = 10, bandcount2 = 10, bandcount3 = 200)
```

`predictRecurrentCount` *Predict the number of recurrent events in a window, by rolling episodes forward*

Description

Experimental. For a patient at the start of (or partway through) an episode of a gap-time recurrent-event model fit with `buildRecurrentEpisodes`, predicts the distribution of the *number* of recurrences in `(prediction_time, prediction_time + horizon]`, together with the probability of dying in it, by Monte Carlo: every patient is simulated forward `n_sim` times, episode after episode, until the window closes or the patient dies. `predictRisk()` answers "is the *next* event a recurrence or a terminal event, and when"; this answers "how many recurrences".

Usage

```
predictRecurrentCount(
  data_predict,
  long_fit_all,
  survival_fit_all,
  prediction_time,
  horizon,
  time_variable,
  survival_variable_all = list(),
  survival_trans_function = list(),
  episode_variable = "episode",
  next_covariates = NULL,
  stay_lengths = 0,
  n_sim = 200,
  max_stages = 10,
  bandcount2 = 100,
  seed = NULL
)
```

Arguments

`data_predict`	The rows of the patients' *open* episode, as `predictRisk()` takes them (for example the `long_data` rows of `buildRecurrentEpisodes()` for each patient's last episode, whose gap is `NA`): one `data.frame` used for every biomarker, or a list of data frames, one per biomarker, as `buildRecurrentEpisodes()` returns

for a list `data_long`; one patient per `id`: more than one episode for the same patient is an error. A patient whose `gap` is not `NA` and lies below `prediction_time` counts as no longer at risk, and is an error too rather than a silent zero. The patient's last row (in the first data frame that has the patient) supplies the covariates of later episodes.

`long_fit_all, survival_fit_all`
: Fits from `longitudinalSub` and `survivalSub`, on episode data.

`prediction_time`
: The time (on the episode clock, as in `predictRisk()`) the prediction is made at: `0` at discharge.

`horizon` Length of the window after `prediction_time`, in the same time unit.

`time_variable` Name of the time variable of the longitudinal sub-model (e.g. `"time_in_episode"`).

`survival_variable_all, survival_trans_function`
: As in `predictRisk`.

`episode_variable`
: Name of the episode-counter covariate (default `"episode"`); used only by the default `next_covariates`, and ignored if the data has no such column.

`next_covariates`
: Optional `function(row, k)` returning the one-row covariate data frame of the episode `k` episodes after the open one (`k = 1, 2, ...`), given the patient's last row of `data_predict` as `row`. The default adds `k` to `episode_variable`, but not beyond the largest episode number in the data the survival sub-model was fit on: past it the fitted effect of the counter is an extrapolation (a hazard that grows with each earlier recurrence would be multiplied a hundredfold or more by episode 30), and the simulated paths would run on through ever shorter episodes.

`stay_lengths` A numeric vector of non-negative stay lengths (the time between a recurrence and the discharge that restarts the process), resampled with replacement. `0` (default): none.

`n_sim` Paths simulated per patient.

`max_stages` Most episodes drawn per path (including the first); paths still running at the cap are counted as they stand, with a warning.

`bandcount2` Event-time grid size passed to `simulateTrajectory`.

`seed` Optional integer seed; the caller's random number stream is left unchanged.

Details

Each simulated path goes as follows.

1. The *current* (open) episode's event time and type are drawn with `simulateTrajectory`, from their posterior given the patient's biomarker history in that episode – the same distribution `predictRisk` integrates, so the first step uses the biomarkers exactly as `predictRisk()` does.
2. If the event is a terminal event inside the window, the path ends (the patient dies). If it is a recurrence inside the window, the count goes up by one; a stay length is then drawn (from `stay_lengths`) and the next episode starts at discharge. If the event (or the discharge) falls after the window closes, the path ends.

3. Each *later* episode is drawn the same way, but with no biomarker history – the patient has not been observed in it yet – so from the model's prior given the patient's covariates, with the episode counter advanced by `next_covariates`. Repeat until no path is still running, or `max_stages` episodes have been drawn.

Limitation. Only the first step uses the patient's biomarkers. Later episodes depend on the patient only through their covariates (and the episode counter), so any patient-level tendency to recur that is not in those covariates – a frailty – is not carried forward, and the predicted spread of the counts (e.g. the chance of two or more recurrences) will be too small when such a tendency is present. Putting the number of previous episodes, or the length of the previous gap, among the survival sub-model's covariates (and updating them in `next_covariates`) recovers some of it.

The stay length is not part of the backward joint model, so it has to be supplied. The default, `0`, means a recurrence takes no time (a tumour recurrence, say); for hospitalizations pass observed stay lengths (they are resampled with replacement).

Value

An object of class `"predictRecurrentCount.BJM"`, a list with

summary One row per patient: `mean_count` (expected number of recurrences in the window), `p_count_0`, `p_count_1`, `p_count_2`, `p_count_3plus`, `p_ge1`, `p_ge2`, and `p_death` (probability of dying in the window, outside hospital: a death during a stay is not part of the model, see `buildRecurrentEpisodes`).

counts The raw draws: a patients x `n_sim` matrix of recurrence counts.

died A patients x `n_sim` logical matrix: died in the window.

horizon, prediction_time, n_sim As supplied.

stages Most episodes drawn on any path.

capped `TRUE` if some path was still running when `max_stages` stopped it (a warning, giving their share, is also given): its later recurrences are not counted, so the counts are slightly low.

capped_fraction The share of all simulated paths stopped by `max_stages` (`0` if none).

See Also

`buildRecurrentEpisodes`, `predictRisk`, `simulateTrajectory`

Examples

```
## Simulated recurrent events and a biomarker. Subject 9999 is still at
## risk in their 2nd episode, and is who we predict for.
set.seed(1)
n <- 60
data_events <- do.call(rbind, lapply(seq_len(n), function(id) {
  k <- sample(1:3, 1)
  data.frame(
    id = id, time = cumsum(rexp(k, rate = 1)),
    status = c(rep(1, k - 1), sample(c(0, 2), 1, prob = c(0.6, 0.4))),
    age = round(50 + rnorm(1, sd = 8)), sex = id %% 2
  )
}))
```

```
data_events <- rbind(data_events, data.frame(
  id = 9999, time = c(0.8, 1.6), status = c(1, 0), age = 55, sex = 1
))
data_long <- do.call(rbind, lapply(c(seq_len(n), 9999), function(id) {
  tt <- seq(0, max(data_events$time[data_events$id == id]) + 0.2, by = 0.25)
  data.frame(id = id, year = tt, biomarker = rnorm(length(tt)))
}))
built <- buildRecurrentEpisodes(data_events, data_long, time_variable_long = "year")

survival_fit_recur <- survivalSub(
  built$episode_data,
  Surv(gap, event_any) ~ age + sex + episode,
  event_type ~ gap + age + sex + episode
)
long_fit_recur <- longitudinalSub(
  built$long_data[built$long_data$event_any == 1, ],
  biomarker ~ time_in_episode + age + sex + gap + gap * time_in_episode,
  ~ time_in_episode | episode_id
)

## The number of recurrences over the next 3 time units
open <- built$long_data[built$long_data$id == 9999 & built$long_data$event_any == 0, ]
counts <- predictRecurrentCount(
  open, long_fit_recur, survival_fit_recur,
  prediction_time = max(open$time_in_episode), horizon = 3,
  time_variable = "time_in_episode", n_sim = 50, bandcount2 = 30, seed = 1
)
counts
```

`predictRisk` *Dynamic prediction function for future event risk*

Description

Combines a fitted longitudinal sub-model (`longitudinalSub`) and survival sub-model (`survivalSub`) into a backward joint model prediction: for each at-risk subject at `prediction_time`, the subject's observed longitudinal history up to that time is used to update their individual random effects (empirical Bayes), which are then integrated against the survival sub-model's hazard to give the predicted probability of experiencing the event within `(prediction_time, prediction_time + horizon]`, conditional on being event-free at `prediction_time`. The numerator (event within the window) is evaluated on a grid over the window whose resolution is controlled by `bandcount1`; the denominator (survival to `prediction_time`, used for normalization) is that same window integral plus the integral beyond the window, evaluated on a grid controlled by `bandcount2`; see Details.

The prediction is conditional on the longitudinal history observed up to `prediction_time`: rows of `data_predict_all` whose `time_variable` is later than `prediction_time` are dropped, with a warning, before predicting.

A patient with no biomarker measurement up to `prediction_time` (rows whose biomarker values are all `NA`, or only rows after `prediction_time`, e.g. a `prediction_time` before the first visit)

has no history to condition on. The longitudinal factor $f(Y \mid T)$ of the model is then 1, and the patient's risk is that of the survival sub-model alone, given the covariates in their earliest row; a warning names these patients. The other patients' predictions do not change. A patient with measurements of some biomarkers but none of another is still not predicted (NA, with a warning).

Usage

```
predictRisk(
  data_predict_all,
  long_fit_all,
  survival_fit_all,
  prediction_time,
  horizon,
  time_variable,
  survival_variable_all,
  survival_trans_function,
  bandcount1 = "auto",
  bandcount2 = "auto",
  n_cores = 1
)
```

Arguments

`data_predict_all`
: This involves a collection of `data.frame` objects for dynamic prediction, each corresponding to a distinct longitudinal outcome. These data frames should contain the variables specified in `long_sub_fixed` and `long_sub_random`. Utilizing a list structure allows for the incorporation of multiple longitudinal outcomes, each potentially following different measurement protocols. In instances where all longitudinal outcomes are recorded at identical time points across patients, a singular `data.frame` object may be used in a `list`. Alternatively, a single bare `data.frame` (not wrapped in a list) may be supplied directly; it is then reused for every longitudinal outcome. It is presumed that each data frame is structured in a long format.

`long_fit_all`
: Outputs from the model fitting process using the `nlme` package, encompassing the results and parameters obtained from the analysis.

`survival_fit_all`
: Results and parameters generated from the model fitting procedure, utilizing the `coxph` function. These outputs include the comprehensive findings and variables derived from the analysis.

`prediction_time`
: Time used to make the prediction.

`horizon`
: Prediction horizon: the length of the window after `prediction_time`, `>= 0`.

`time_variable`
: The name of time variable in linear mixed model.

`survival_variable_all`
: The name of the transformed time-to-event outcomes variable. The longitudinal sub-model must use the event time, directly or through one of these transformations.

survival_trans_function
: The transformation function used for time-to-event outcomes, in the order of survival_variable_all. They are checked against the columns the longitudinal sub-model was fit on, and a mismatch (e.g. two functions in the wrong order) is an error.

bandcount1
: The number of grid points spanning the prediction window, from prediction_time to prediction_time + horizon (the numerator of the risk probability). Larger values give a more accurate but slower estimate. Defaults to "auto" (see Details).

bandcount2
: The number of grid points spanning [prediction_time + horizon, upper_bound], where upper_bound is set internally as the earliest time by which every at-risk patient's model-based probability of still being event-free (given event-free at prediction_time + horizon) has dropped below 1e-4; this approximates integrating out to infinity for the part of the denominator beyond the prediction window. A wider follow-up range needs a larger bandcount2 to keep the grid spacing comparable. Defaults to "auto" (see Details).

n_cores
: Number of CPU cores. With an ordinal biomarker the prediction's cost is per patient, and the patients are spread over n_cores cores with parallel::mclapply(); this is only parallel on Unix-like systems (Linux, macOS), and runs serially on Windows. The result is exactly the same for any n_cores. With continuous biomarkers only, predictions are fast and n_cores has no effect. Defaults to 1.

Details

There is no universal correct value for bandcount1/bandcount2: as a practical check, double both and confirm the resulting risk probabilities barely change; if they do, keep doubling. By default (bandcount1 = "auto", bandcount2 = "auto"), this doubling check is done for you: starting from small built-in values, both are doubled together, and the result is compared to the previous round, until the largest relative change in the risk probabilities drops below 1%, or 2 doublings have been tried (so at most 3 calls' worth of work). If it still has not converged by then, a warning reports this and the result at the largest value tried is returned anyway (not an error), so this never silently loops for an unbounded amount of time. Pass an explicit number for either argument to skip auto-tuning it and use a fixed value instead (as in previous package versions), or call checkBandcountConvergence() directly for more control over the tolerance and doubling count. See also vignette("BJM-intro",package = "BJM") for a worked example.

The denominator integrates over every event time after prediction_time, including times beyond the last follow-up time in the data survivalSub() was fit on. There, the baseline cumulative hazard is extrapolated linearly (a constant hazard), and the longitudinal sub-model's mean is evaluated at event times it was never fit on. When much of an at-risk patient's survival probability lies beyond the last follow-up time (e.g. a prediction late in follow-up), the prediction depends on this extrapolation; in pbc3 it moved the risks checked by less than one percentage point.

For a fit with an ordinal biomarker, a patient's ordinal measurements contribute the probability of their observed categories. With a single ordinal random effect (e.g. one ordinal biomarker with a random intercept) it is computed by Gauss-Hermite quadrature, and is exact; with more, by Monte Carlo (mvtnorm::pmvnorm()), with an error around the fourth significant digit. Each patient's Monte Carlo draws use a fixed seed derived from the patient's id, so the results are the same on every run and for any n_cores, without set.seed(), and the caller's random-number state is left unchanged.

Value

An object of class "predictRisk.BJM", a named list with elements:

risk_prob_1 A vector of dynamically predicted probabilities, one per at-risk patient, of experiencing the (first) event within the prediction horizon, named by patient id. All 0 when horizon = 0.

risk_prob_2 When survival_fit_all was fit with competing risks, a vector of dynamically predicted probabilities, one per at-risk patient, of experiencing the competing event within the prediction horizon, named by patient id (all 0 when horizon = 0). NULL when there is no competing risk.

prob_undetected_1, prob_undetected_2 Only for an interval-censored survival_fit_all (see below): the probability that the (first / competing) event already happened between the patient's last visit and prediction_time, undetected. prob_undetected_2 is NULL without competing risks.

Interval-censored fits. An event is then only detected at a visit, so at prediction_time s a patient is known to be event-free only up to their last visit V (the last time in their history up to s). risk_prob_1 is then $P(s < T \leq s + horizon \mid T > V, history)$ and prob_undetected_1 is $P(V < T \leq s \mid T > V, history)$ (with competing risks, each _1/_2 value is that probability jointly with the event type); with $V = s$ they reduce to the usual conditioning on $T > s$. Only patients still at risk at prediction_time are predicted: a patient whose recorded survival time is before prediction_time is left out (one with a missing survival time is kept). The names show which patients each value belongs to. The numerical integration grids are shared by all patients predicted in one call, so a patient's value can differ slightly (within the grid's discretization error, which shrinks as bandcount1/bandcount2 grow) depending on which other patients are predicted alongside it.

Examples

```
data(pbc3)

data_survival_fitting =  pbc3[!duplicated(pbc3$id), ]

form_marginal_surv = Surv(years, status3) ~ age + sex
form_conditional_cr = NULL

survival_fit_all = survivalSub(data_survival_fitting, form_marginal_surv,
                               form_conditional_cr)

long_sub_fixed = list(
  "long1" = serBilir ~ year + age + sex +  (years) + (years) * year,
  "long2" = prothrombin ~ year + age + sex + (years) + (years) * year,
  "long3" = albumin ~ year + age + age * year + sex + (years) + (years) * year,
  "long4" = alkaline ~ year + age + sex + (years) + (years) * year,
  "long5" = SGOT ~ year + age + sex + (years) + (years) * year,
  "long6" = platelets ~ year + age + sex + (years)  + (years) * year)

long_sub_random =list(
  "long1" =  ~ year| id,
```

```
  "long2" =  ~ year| id,
  "long3" =  ~ year| id,
  "long4" =  ~ year| id,
  "long5" =  ~ year| id,
  "long6" =  ~ year| id)

survival_variable_all = list(
  "Tyears1",  "Tyears2", "Tyears3", "Tyears4"
)

survival_trans_function = list(
  fun1 = function(x){abs(x - 1)},
  fun2 = function(x){abs(x - 3)},
  fun3 = function(x){abs(x - 5)},
  fun4 = function(x){abs(x - 7)}
)

# Complete case analysis
data_fit_all = list()
for(i in seq_len(length(long_sub_fixed))){
  data_fit_all[[i]] = pbc3[pbc3$status3 == 1, ]
}

# fitting longitudinal submodel
long_fit_all = longitudinalSub(data_fit_all, long_sub_fixed, long_sub_random)

i_PID = 2
data.raw.predict.1 = pbc3[pbc3$id == i_PID, ]

data_predict_all = list()
for(i in seq_len(length(long_sub_fixed))){
  data_predict_all[[i]] = data.raw.predict.1[data.raw.predict.1$year <= 3,]
}

# predict risk probability
risk.prob = predictRisk(data_predict_all, long_fit_all, survival_fit_all,
                              prediction_time = 3,
                              horizon = 3, time_variable = "year",
                              survival_variable_all, survival_trans_function,
                              bandcount1 = 10, bandcount2 = 10)

# poly() in its default orthogonal mode, splines::ns()/bs(), and factor()
# in a continuous biomarker's long_sub_fixed produce correct dynamic
# predictions (see ?longitudinalSub) -- including
# when a patient has only a single longitudinal observation to condition
# on -- because the basis/contrasts fit on the full training data are
# cached (via each biomarker's terms object and long_fit_all$xlevels) and
# reused here at prediction time, instead of being recomputed from that
# patient's small per-prediction-time slice:
long_sub_fixed_nonlinear = list(
  "long1" = serBilir ~ poly(year, 2) + age + sex + years,
  "long2" = albumin ~ year + age + sex + years)
long_sub_random_nonlinear = list("long1" = ~ year | id, "long2" = ~ year | id)
```

```
long_fit_nonlinear = longitudinalSub(pbc3[pbc3$status3 == 1, ],
                                     long_sub_fixed_nonlinear,
                                     long_sub_random_nonlinear)

data_predict_normal = data.raw.predict.1[data.raw.predict.1$year <= 3, ]
data_predict_sparse = data.raw.predict.1[1, ]

risk.prob.normal = predictRisk(data_predict_normal, long_fit_nonlinear,
                                     survival_fit_all, prediction_time = 3,
                                     horizon = 3, time_variable = "year",
                                     survival_variable_all, survival_trans_function,
                                     bandcount1 = 10, bandcount2 = 10)
risk.prob.sparse = predictRisk(data_predict_sparse, long_fit_nonlinear,
                                     survival_fit_all, prediction_time = 3,
                                     horizon = 3, time_variable = "year",
                                     survival_variable_all, survival_trans_function,
                                     bandcount1 = 10, bandcount2 = 10)
# both give a sane, non-degenerate risk_prob_1 (not 0, no error)
risk.prob.normal$risk_prob_1
risk.prob.sparse$risk_prob_1
```

`riskPlot` *Plot predicted risk across a sweep of landmark times*

Description

Fixes the prediction horizon and sweeps backward/forward over a set of landmark times, calling `predictRisk` at each landmark in `prediction_time` to trace out how predicted event risk over that fixed-length window changes as the landmark moves – i.e. as more (or less) of the subject's longitudinal history is used to update their risk. `prediction_time` may be a vector of landmark times to use directly; a single value, which is expanded to three landmarks (`prediction_time`, `1.5 * prediction_time`, `2 * prediction_time`); or `NULL`, which uses every observed longitudinal measurement time (across subjects, from the first biomarker's data) as a landmark. This contrasts with `predictPlot`, which instead fixes the landmark and sweeps over a range of horizons. The optional `bio_i` argument only selects which biomarker's observed trajectory is overlaid on the plot for visual reference – it does not affect the risk computation itself (unlike `predictPlot`'s `bio_pred`, which drives a biomarker density prediction).

When the data give the subject's survival time and status, a vertical line marks the end of follow-up: solid for an observed event (coloured by event type with competing risks), dashed grey for a censoring.

Usage

```
riskPlot(
  data_predict_all_pre,
```

```
  long_fit_all,
  survival_fit_all,
  prediction_time = NULL,
  bio_i = NULL,
  horizon,
  time_variable,
  survival_variable_all,
  survival_trans_function,
  bandcount1 = "auto",
  bandcount2 = "auto",
  n_cores = 1
)
```

Arguments

`data_predict_all_pre`
: This involves a collection of `data.frame` objects for dynamic prediction, each corresponding to a distinct longitudinal outcome. These data frames should contain the variables specified in `long_sub_fixed` and `long_sub_random`. Utilizing a list structure allows for the incorporation of multiple longitudinal outcomes, each potentially following different measurement protocols. In instances where all longitudinal outcomes are recorded at identical time points across patients, a singular `data.frame` object may be used in a `list`. Alternatively, a single bare `data.frame` (not wrapped in a list) may be supplied directly; it is then reused for every longitudinal outcome. It is presumed that each data frame is structured in a long format.

`long_fit_all`
: Outputs from the model fitting process using the `nlme` package, encompassing the results and parameters obtained from the analysis.

`survival_fit_all`
: Results and parameters generated from the model fitting procedure, utilizing the `coxph` function. These outputs include the comprehensive findings and variables derived from the analysis.

`prediction_time`
: The landmark time(s) at which risks are predicted and plotted: `NULL` (the default) uses every visit time (`time_variable`) in the patient's data; a single value `s` uses the three landmarks `s`, `1.5 * s` and `2 * s`; a vector uses those times. At each landmark only the history up to it is used.

`bio_i`
: Biomarker whose observed history is drawn alongside the predicted risks (by position in `long_fit_all`); `NULL` draws the first one.

`horizon`
: Prediction horizon.

`time_variable`
: The name of time variable in linear mixed model.

`survival_variable_all`
: The name of the transformed time-to-event outcomes variable.

`survival_trans_function`
: The transformation function used for time-to-event outcomes, in the order of `survival_variable_all`.

bandcount1 — The number of grid points spanning the prediction window, from prediction_time to prediction_time + horizon. Larger values give a more accurate but slower estimate. Defaults to "auto", which resolves it once, before looping over the landmark times, (using the first landmark time as a representative probe) by doubling from a built-in starting value until the predicted risk stabilizes; see predictRisk's bandcount1 for details of that search. The resolved value is then reused, fixed, for every landmark time – it is not re-searched on every iteration.

bandcount2 — The number of grid points used to approximate integrating out to infinity when normalizing the predicted risk. A wider follow-up range needs a larger bandcount2 to keep the grid spacing comparable. Defaults to "auto"; resolved the same way as bandcount1 (jointly with it, when both are "auto").

Pass explicit numbers instead of "auto" for full manual control, or use checkBandcountConvergence() (applied to predictRisk() directly) to inspect the convergence behavior yourself. See also vignette("BJM-intro", package = "BJM") for further guidance on choosing bandcount1/bandcount2.

n_cores — Number of CPU cores to use for computing the prediction at each landmark time in prediction_time. Each landmark time is computed independently, so this loop can be dispatched across cores. Defaults to 1 (serial execution; identical behavior/output to versions of this function without this argument). Values greater than 1 use parallel::mclapply(), which relies on forking and is therefore only actually parallel on Unix-like systems (Linux, macOS); on Windows, mclapply() silently runs the iterations serially regardless of n_cores (a limitation of R's fork-based parallelism, not of this package). Parallel execution produces exactly the same numeric result as serial execution – only the order in which iterations are computed (not the order results are assembled in) changes.

Value

Plot of risk using dynamic prediction.

Examples

```
data(pbc3)
survival_fit_all <- survivalSub(pbc3[!duplicated(pbc3$id), ],
                                Surv(years, status3) ~ age + sex, NULL)
long_fit_all <- longitudinalSub(pbc3[pbc3$status3 == 1, ],
                                list(serBilir ~ year + age + sex + years,
                                     albumin ~ year + age + sex + years),
                                list(~ year | id, ~ year | id))

# one patient's whole history; risks within 2 years of landmarks 2, 3, 4
patient <- pbc3[pbc3$id == 2, ]
riskPlot(patient, long_fit_all, survival_fit_all, prediction_time = c(2, 3, 4),
         bio_i = 1, horizon = 2, time_variable = "year",
         survival_variable_all = list(), survival_trans_function = list(),
         bandcount1 = 10, bandcount2 = 20)
```

`simulateTrajectory` *Simulate future trajectories from the backward joint model*

Description

Draws `n_sim` complete futures for each patient from the fitted backward joint model, conditional on the patient's longitudinal history up to `prediction_time` and on being event-free at `prediction_time`: an event time (and, with competing risks, an event type), and the values of every biomarker at the requested `times`. Where `predictRisk` and `predictLongitudinal` summarize the predictive distribution, this returns samples from it, so it can be used to generate synthetic patients or to propagate the prediction into any other quantity (e.g. a patient's "digital twin" under the fitted model).

Each draw follows the model's factorization $f(T, D)\, f(Y \mid T, D)$:

1. (T, D) is drawn from its posterior given the history, proportional to $f(Y_{hist} \mid T, D)\, f(D \mid T)\, f(T)$ restricted to $T >$ `prediction_time`, evaluated on the same integration grid as `predictRisk`'s denominator;
2. the random effects b are drawn from their Gaussian posterior given the history and the drawn (T, D);
3. the biomarkers at `times` are drawn from the longitudinal sub-model given (T, D, b), adding residual error.

A patient with no non-missing biomarker measurement up to `prediction_time` (e.g. a single baseline row with the biomarkers set to `NA`) is simulated from the model's prior given their baseline covariates, i.e. as a new synthetic patient.

Usage

```
simulateTrajectory(
  data_predict_all,
  long_fit_all,
  survival_fit_all,
  prediction_time,
  times,
  time_variable,
  survival_variable_all,
  survival_trans_function,
  n_sim = 100,
  bandcount2 = 100,
  max_event_time = NULL,
  truncate = TRUE,
  future_covariates = NULL,
  seed = NULL
)
```

Arguments

`data_predict_all`
: This involves a collection of `data.frame` objects for dynamic prediction, each corresponding to a distinct longitudinal outcome. These data frames should contain the variables specified in `long_sub_fixed` and `long_sub_random`. Utilizing a list structure allows for the incorporation of multiple longitudinal outcomes, each potentially following different measurement protocols. In instances where all longitudinal outcomes are recorded at identical time points across patients, a singular `data.frame` object may be used in a `list`. Alternatively, a single bare `data.frame` (not wrapped in a list) may be supplied directly; it is then reused for every longitudinal outcome. It is presumed that each data frame is structured in a long format.

`long_fit_all`
: Outputs from the model fitting process using the `nlme` package, encompassing the results and parameters obtained from the analysis.

`survival_fit_all`
: Results and parameters generated from the model fitting procedure, utilizing the `coxph` function. These outputs include the comprehensive findings and variables derived from the analysis.

`prediction_time`
: Time used to make the prediction.

`times`
: Numeric vector of times (on the scale of `time_variable`) at which to simulate the biomarkers, each at least `prediction_time`.

`time_variable`
: The name of time variable in linear mixed model.

`survival_variable_all`
: The name of the transformed time-to-event outcomes variable. The longitudinal sub-model must use the event time, directly or through one of these transformations.

`survival_trans_function`
: The transformation function used for time-to-event outcomes, in the order of `survival_variable_all`. They are checked against the columns the longitudinal sub-model was fit on, and a mismatch (e.g. two functions in the wrong order) is an error.

`n_sim`
: Number of futures to draw per patient.

`bandcount2`
: The number of intervals the event-time distribution after `prediction_time` is discretized into (see Details and `predictRisk`).

`max_event_time`
: Event times beyond this are reported as event-free through it, and the model is evaluated at it in their place (see Details). Defaults to `NULL`, the last follow-up time of the data `survivalSub()` was fit on; `Inf` turns this off.

`truncate`
: If `TRUE`, biomarker values at or after the drawn event time are `NA` (see Details).

`future_covariates`
: Optional data frame of covariate values at future times: a `time_variable` column, one column per covariate to set, and optionally the patient id column (named as in `long_sub_random`) to set them per patient; without it, every row applies to every patient. It cannot set the biomarkers or the event time or type. See Details.

`seed`
: Optional integer seed, for reproducible draws. The caller's random number stream is left unchanged.

Details

The event time is drawn on the grid of `bandcount2` intervals that `predictRisk` integrates over (each holding about the same share of the conditional survival probability, with an extra edge at `max_event_time`): an interval is drawn with its posterior probability, the model is evaluated at its midpoint, and the event time reported is uniform within it, so that summaries of the drawn event times (e.g. the share before a given time) are not biased by rounding to the midpoints.

Past the last follow-up time of the data `survivalSub()` was fit on, the event-time distribution rests on an extrapolated baseline hazard and the longitudinal sub-model's mean on event times it was never fit on; drawn from a patient's prior (no history), a large share of event times can land there, with implausible biomarker values. So an event time beyond `max_event_time` (by default that last follow-up time) is reported as event-free through `max_event_time` (`event = 0`, `event_time = max_event_time`), and both the history's likelihood and the simulated biomarkers use `T = max_event_time` in its place, i.e. a patient whose event comes after follow-up is assumed to follow the trajectory of one whose event comes at its end. `max_event_time = Inf` turns this off and reproduces the event-time distribution `predictRisk` integrates over.

The biomarkers at `times` are evaluated with the covariates of the patient's last row in `data_predict_all` (with `time_variable` set to each element of `times`), so a time-varying covariate is carried forward at its last value, unless `future_covariates` says otherwise. Each row of `future_covariates` sets the covariates in its columns from its `time_variable` on, until a later row changes them: a time in `times` takes the covariates of the last row at or before it, and those of the patient's last row in `data_predict_all` for anything that row does not set. The history, and so the posterior of the event time and random effects, is unaffected. A covariate that is a known function of time (such as current age) is better written into the model formula as one (e.g. `I(age + year)`), so that it is updated without `future_covariates`. Comparing draws under different `future_covariates` compares the outcomes the model associates with those covariate paths; it is not a causal effect of changing them.

The backward model describes the biomarkers before the event. With `truncate = TRUE` (the default) a value at a time at or after the drawn (or censored) event time is `NA`; with `truncate = FALSE` it is still drawn from the sub-model, which is the quantity whose density `predictLongitudinal` returns.

An ordinal biomarker (see `longitudinalSubCopula`) is the discretization of a latent Gaussian score by the fitted thresholds, and is simulated that way: its latent score is drawn like a continuous biomarker (with unit residual variance) and cut into categories. In the history it is known only to have fallen between the thresholds bracketing its observed category: its contribution to the likelihood of (T, D) is a multivariate normal box probability, computed as in `predictRisk`, and the random effects are drawn by first drawing the history's latent scores from their truncated multivariate normal distribution given the continuous measurements (by rejection, or by Gibbs sampling when few proposals fall in the box), and then the random effects given those scores. Supply `seed` for reproducible draws.

Value

A data frame of class `"simulateTrajectory.BJM"`, with one row per patient, draw and element of `times`, and columns: the patient id (named as in `long_sub_random`); `sim`, the draw number; `time_variable`; `event_time`, the drawn event time, or `max_event_time` if the event comes after it; `event`, 1 if the event occurs at `event_time` and 0 if the patient is event-free through it; with competing risks, the drawn event type (named as the response of `form_conditional_cr`, with the same 0/1 coding, and `NA` when `event` is 0); and one column per biomarker, named by its response

variable (an ordered factor, with the fitted categories as levels, for an ordinal biomarker). Attributes `"prediction_time"`, `"times"` and `"max_event_time"` record the values used, and `"history"` the continuous biomarker measurements conditioned on (for `plot.simulateTrajectory.BJM`).

See Also

`plot.simulateTrajectory.BJM` to plot the draws.

Examples

```
data(pbc3)

survival_fit_all = survivalSub(pbc3[!duplicated(pbc3$id), ],
                               Surv(years, status3) ~ age + sex, NULL)

long_sub_fixed = list(
  "long1" = serBilir ~ year + age + sex + (years) + (years) * year,
  "long2" = albumin ~ year + age + sex + (years) + (years) * year)
long_sub_random = list("long1" = ~ year | id, "long2" = ~ year | id)
data_fit_all = list(pbc3[pbc3$status3 == 1, ], pbc3[pbc3$status3 == 1, ])
long_fit_all = longitudinalSub(data_fit_all, long_sub_fixed, long_sub_random)

trans = survivalTrans(c(1, 3, 5, 7))

# Patient 2's history up to year 3: 200 possible futures over years 3-8
history = pbc3[pbc3$id == 2 & pbc3$year <= 3, ]
sims = simulateTrajectory(list(history, history), long_fit_all, survival_fit_all,
                          prediction_time = 3, times = seq(3, 8, by = 0.5),
                          time_variable = "year", trans$survival_variable_all,
                          trans$survival_trans_function, n_sim = 200, seed = 1)
head(sims)

# Covariates after the last row are carried forward unless
# future_covariates sets them from a given time on. Age at entry does not
# change, so this is only a sensitivity check of the forecast's dependence
# on age; a time-varying covariate such as ascites would be set the same
# way, e.g. data.frame(year = 6, ascites = "Yes").
sims_older = simulateTrajectory(list(history, history), long_fit_all, survival_fit_all,
                                prediction_time = 3, times = seq(3, 8, by = 0.5),
                                time_variable = "year", trans$survival_variable_all,
                                trans$survival_trans_function, n_sim = 200, seed = 1,
                        future_covariates = data.frame(year = 5, age = history$age[1] + 10))
tapply(sims_older$serBilir - sims$serBilir, sims$year, mean, na.rm = TRUE)

# Probability of an event within 2 years, from the draws
first = sims[!duplicated(sims$sim), ]
mean(first$event == 1 & first$event_time <= 5)

# A new synthetic patient: baseline covariates only, biomarkers NA
new_patient = data.frame(id = "new", year = 0, age = 50, sex = 1,
                         serBilir = NA, albumin = NA)
synthetic = simulateTrajectory(new_patient, long_fit_all, survival_fit_all,
```

```
                                prediction_time = 0, times = 0:6, time_variable = "year",
                                trans$survival_variable_all,
                                trans$survival_trans_function, n_sim = 100, seed = 1)

plot(sims)
plot(sims, which = "event")
```

---

spaghettiPlot *Plot individual longitudinal trajectories (spaghetti plot)*

Description

Draws one line per subject for a longitudinal biomarker, optionally colored by the subject's eventual outcome, with a smoothed mean trajectory per outcome group overlaid. With `align = "event"` the horizontal axis becomes the time remaining until the subject's event/censoring time (`survival_variable - time_variable`, so 0 is the event time, at the left edge). This mirrors the backward joint model's view of the biomarker as a function of time-to-event. Unlike `cmtPlot`, which averages over subjects sharing one event time, this shows every subject's raw history.

Usage

```
spaghettiPlot(
  data_plot_all,
  bio_variable,
  time_variable,
  id_variable = "id",
  survival_variable = NULL,
  event_type_variable = NULL,
  align = c("baseline", "event"),
  n_subjects = NULL,
  smooth = TRUE,
  alpha = 0.3
)
```

Arguments

`data_plot_all`	A long-format `data.frame` with one row per biomarker measurement.
`bio_variable`	Name of the biomarker variable to plot.
`time_variable`	Name of the measurement-time variable.
`id_variable`	Name of the subject ID column. Default is `"id"`.
`survival_variable`	Name of the time-to-event variable. Required when `align = "event"`; otherwise ignored.

event_type_variable
: Name of the variable used to color subjects (e.g. the event indicator or event type). Must be constant within each subject. NULL (default) draws every subject in one color.

align
: Either "baseline" (default), plotting against time_variable, or "event", plotting against the time until the event, survival_variable - time_variable.

n_subjects
: Optional number of subjects to draw, sampled at random (call set.seed() first for a reproducible sample). NULL (default) draws every subject. The smoothed means use the drawn subjects only.

smooth
: Logical; overlay a LOESS mean trajectory per group. Default TRUE.

alpha
: Transparency of the individual lines. Default 0.3.

Value

A ggplot object.

Examples

```
data(pbc3)

# trajectories colored by death (1) vs alive/transplanted (0)
spaghettiPlot(pbc3, bio_variable = "serBilir", time_variable = "year",
  event_type_variable = "status2")

# aligned at each subject's event/censoring time
spaghettiPlot(pbc3, bio_variable = "serBilir", time_variable = "year",
  survival_variable = "years", event_type_variable = "status2",
  align = "event")
```

---

survivalSub *Fit the survival sub-model*

Description

Fits the survival building block used by predictRisk and dynamicPredictionBio: a marginal Cox proportional-hazards model (via coxph, with x = TRUE, y = TRUE so the fit is self-contained for later prediction) for the time-to-event outcome given form_marginal_surv, plus, optionally, a logistic event-type/competing-risks model (via glm with family = binomial) given form_conditional_cr. The competing-risks model is fit only among subjects who experienced an event (i.e. whose censoring indicator is non-zero), predicting which type of event occurred conditional on an event having occurred. Together with longitudinalSub, the object returned here forms the pair of sub-models that dynamic prediction is built on.

Usage

```
survivalSub(
  data_survival_fitting,
  form_marginal_surv,
  form_conditional_cr,
  event_time = NULL,
  baseline = c("spline", "piecewise"),
  df = NULL
)
```

Arguments

data_survival_fitting	One row per subject, containing the time-to-event outcome, censoring/event-type indicator, and any baseline covariates referenced in form_marginal_surv or form_conditional_cr.
form_marginal_surv	A survival formula, e.g. Surv(time, status) ~ covariates, passed to coxph to fit the marginal event-time model.
form_conditional_cr	An optional formula for the competing-risks (event-type) model, e.g. event_type ~ covariates, passed to glm with family = binomial. Set to NULL when there is only a single event type (no competing risks).
event_time	Only for an interval-censored outcome: the name of the event-time column that the longitudinal sub-model's formulas use (the exact time is unknown, so it is not part of form_marginal_surv).
baseline	Only for an interval-censored outcome: the baseline hazard, "spline" (default; Royston–Parmar spline for the log cumulative hazard in log time) or "piecewise" (piecewise-constant hazard).
df	Only for an interval-censored outcome: spline degrees of freedom (default 3) or number of pieces (default 6).

Details

Interval censoring (experimental). When form_marginal_surv is Surv(L, R, type = "interval2") ~ covariates – the event is only known to lie in (L, R], with R = NA or Inf for right-censored subjects – the marginal model is a proportional hazards model with a smooth spline (or piecewise-constant) baseline hazard fit by maximum likelihood (see icphFit()), stored as ic_fit in place of coxph_fit. A strata() term gives each stratum its own baseline hazard. With competing risks, the event type (0/1) is taken as known once the event is detected, and form_conditional_cr is fit among subjects with an observed event (finite R); if it uses the event time, that column must be filled for them, as fitIntervalBJM does with imputed times. Ordinary right-censored Surv(time, status) outcomes are fit with coxph exactly as before.

Events that happened before a subject's first visit (L = 0) only say that the event fell between time 0 and that visit. Before the earliest visit in the data there are no interval endpoints, so the baseline hazard there is extrapolated, not estimated, and fitIntervalBJM places these subjects' event times according to that extrapolation. print() reports how many events are of this kind: when they are a

large share of all events (in simulations, around a third or more), results can depend noticeably on the assumed shape of the early hazard.

Value

An object of class `"survivalSub.BJM"`, a named list with elements:

coxph_fit The fitted `coxph` marginal survival model (`ic_fit`, an interval-censored PH fit, in its place for an interval-censored outcome).

form_marginal_surv The `form_marginal_surv` formula, as supplied.

glm_fit The fitted `glm` competing-risks (event type) model, or `NULL` if `form_conditional_cr` was not supplied.

form_conditional_cr The `form_conditional_cr` formula, as supplied (or `NULL`).

Examples

```
data(pbc3)

data_survival_fitting =  pbc3[!duplicated(pbc3$id), ]

form_marginal_surv = Surv(years, status3) ~ age + sex
form_conditional_cr = NULL

survival_fit_all = survivalSub(data_survival_fitting, form_marginal_surv,
                               form_conditional_cr)
```

`survivalTrans` *Build a survival-time transform basis from cut points*

Description

`predictRisk()`, `dynamicPredictionBio()`, `predictPlot()`, and `riskPlot()` all take a pair of arguments, `survival_variable_all`/`survival_trans_function`, that describe transformed basis variables of the (remaining) survival time; these can optionally be referenced in `long_sub_fixed` formulas to let the longitudinal sub-model depend flexibly on time-to-event. In every example in this package, that pair follows the same convention: variables named `"Tyears1"`, `"Tyears2"`, ... , each defined as the absolute distance from a fixed cut point (`function(x) abs(x - k)`). `survivalTrans()` builds exactly that pair from a plain vector of cut points, so you do not have to hand-write two parallel lists of matching names and closures. You remain free to construct `survival_variable_all`/`survival_trans_functior` by hand for any other transform.

Usage

```
survivalTrans(cut_points, prefix = "Tyears")
```

Arguments

cut_points	A non-empty numeric vector of cut points, one per transformed basis variable.
prefix	Prefix used for the generated variable names in survival_variable_all ("Tyears" by default, giving "Tyears1", "Tyears2", ...).

Value

A named list with elements survival_variable_all and survival_trans_function, in the format expected by predictRisk(), dynamicPredictionBio(), predictPlot(), and riskPlot().

Examples

```
trans <- survivalTrans(c(1, 3, 5, 7))
trans$survival_variable_all
trans$survival_trans_function[[1]](2)

# equivalent to hand-writing:
survival_variable_all <- list("Tyears1", "Tyears2", "Tyears3", "Tyears4")
survival_trans_function <- list(
  fun1 = function(x) abs(x - 1),
  fun2 = function(x) abs(x - 3),
  fun3 = function(x) abs(x - 5),
  fun4 = function(x) abs(x - 7)
)

# survivalTrans() only ever builds the abs(x - k) family above. Any other
# transform -- as long as it is a function of a single numeric time value
# that returns a single finite numeric value -- is written by hand the
# same way, one entry of survival_trans_function per entry of
# survival_variable_all. A few examples that all work equally well with
# predictRisk()/dynamicPredictionBio()/predictPlot()/riskPlot():
survival_variable_all <- list("Tlog", "Tsqrt", "Tsq", "Texp", "Tinv")
survival_trans_function <- list(
  fun1 = function(x) log(x + 1),        # log(x + 1)
  fun2 = function(x) sqrt(abs(x)),      # sqrt(abs(x))
  fun3 = function(x) x^2,               # x^2
  fun4 = function(x) exp(-x / 10),      # exp(-x / 10)
  fun5 = function(x) 1 / (x + 1)        # 1 / (x + 1)
)
sapply(survival_trans_function, function(f) f(2))
```

Index

∗ **datasets**
pbc3, 26

buildRecurrentEpisodes, 2, *45*, *47*

calibrationPlot, 8, *30*
checkBandcountConvergence, 10, *10*, *29*
cifPlot, 12
cmtPlot, 14, *60*
coxph, *61–63*

dynamicPredictionBio, *23*, *61*

fitIntervalBJM, 16, *62*

glm, *61–63*

imputeLongitudinal, 19, *35*

lme, *23*, *25*
longitudinalSub, *9*, *17*, *20*, *22*, 23, *29–31*, *35*, *37*, *46*, *48*, *61*
longitudinalSubCopula, *37*, *58*

pbc, *27*
pbc3, *3*, *5*, *6*, 26
performancePlot, 28
plot.longitudinalSub.BJM, 30
plot.predictLongitudinal.BJM, 31
plot.simulateTrajectory.BJM, 32, *59*
plot.survivalSub.BJM, 34
poolLongitudinalSub, *17*, *18*, *21*, 35
predictLongitudinal, *21*, *31*, 37, *41*, *56*, *58*
predictPlot, 41, *53*
predictRecurrentCount, *3*, 45
predictRisk, *8–10*, *17*, *21*, *23*, *28*, *29*, *36*, *37*, *41*, *43*, *46*, *47*, 48, *53*, *55–58*, *61*

riskPlot, *42*, 53

simulateTrajectory, *32*, *33*, *46*, *47*, 56

spaghettiPlot, 60
survfit, *12*
survivalSub, *9*, *17*, *18*, *23*, *29*, *34*, *37*, *46*, *48*, 61
survivalTrans, 63