pencal: an R Package for the Dynamic Prediction of Survival with Many Longitudinal Predictors

by Mirko Signorelli

Abstract In survival analysis, longitudinal information on the health status of a patient can be used to dynamically update the predicted probability that a patient will experience an event of interest. Traditional approaches to dynamic prediction such as joint models become computationally unfeasible with more than a handful of longitudinal covariates, warranting the development of methods that can handle a larger number of longitudinal covariates. We introduce the R package **pencal**, which implements a Penalized Regression Calibration approach that makes it possible to handle many longitudinal covariates as predictors of survival. **pencal** uses mixed-effects models to summarize the trajectories of the longitudinal covariates up to a prespecified landmark time, and a penalized Cox model to predict survival based on both baseline covariates and summary measures of the longitudinal covariates. This article illustrates the structure of the R package, provides a step by step example showing how to estimate PRC, compute dynamic predictions of survival and validate performance, and shows how parallelization can be used to significantly reduce computing time.

Introduction

Risk prediction models (Steyerberg, 2009) allow to estimate the probability that an event of interest will occur in the future. Such models are commonly employed in the biomedical field to estimate the probability that an individual will experience an adverse event, and their output can be used to inform patients, monitor their disease progression, and guide treatment decisions.

Traditionally, risk prediction models only used covariate values available at the beginning of the observation period to predict survival. Thus, predictions based on such models could not exploit information gathered at later time points. Because information about the evolution of time-dependent covariates may influence the occurrence of the survival outcome, it is desirable to be able to dynamically update predictions of survival as more longitudinal information becomes available.

Dynamic prediction models employ both baseline and follow-up information to predict survival, and they can be used to update predictions each time new follow-up data are gathered. Three commonly-used statistical methods for dynamic prediction are the time-dependent Cox model, joint modelling of longitudinal and survival data, and landmarking approaches.

The time-dependent Cox model (Therneau and Grambsch, 2000) is an extension of the Cox proportional hazards model that allows for the inclusion of time-dependent covariates. It assumes the value of a time-dependent covariate to be constant between two observation times, and it is only suitable for exogenous time-dependent covariates. The model can be estimated using the R package **survival** (Therneau and Grambsch, 2000).

Joint models for longitudinal and survival outcomes (Henderson et al., 2000) are shared random effects models that combine a submodel for the longitudinal covariates (typically linear mixed models) and one for the survival outcome (usually a Cox model or a parametric survival model). Thanks to the shared random effects formulation, such models are capable to account for a possible interdependence between the longitudinal covariates and the survival outcome. However, the estimation of the shared random effects model is computationally intensive task that has so far restricted the application of joint models to problems with one or few longitudinal covariates. Over the years, several alternative approaches to the estimation of joint models have been proposed, among which are the R packages **JM** (Rizopoulos, 2010), **JMbayes** (Rizopoulos, 2014), **joineR** (Philipson et al., 2018) and **joineRML** (Hickey et al., 2018).

Lastly, landmarking (Van Houwelingen, 2007) is an approach that dynamically adjusts predictions by refitting the prediction model using all subjects that are still at risk at a given landmark time. Landmarking typically involves two modelling steps. In the first step, repeated measurements of the time-dependent covariates up to the landmark time are summarized using either a summary measure or a suitable statistical model. In the second step, the summaries thus computed are used as predictors of survival alongside with the time-independent covariates. The simplest form of landmarking is the last observation carried forward (LOCF) method, which uses the last available measurement of each longitudinal covariate taken up to the landmark time as summary. The main advantage of this approach is that it is easy to implement, it is computationally straightforward and it does not

require the development of dedicated software. Limitations of LOCF landmarking include the fact that it discards all previous repeated measurements, and it does not perform any measurement error correction on the longitudinal covariates (which may be particularly desirable for biomarkers and diagnostic tests that can be subject to measurement error). To overcome these limitations, mixed-effects models can be used to model the trajectories of the longitudinal covariates (Signorelli et al., 2021; Putter and van Houwelingen, 2022; Devaux et al., 2023). While Putter and van Houwelingen (2022) focused on situations with a single longitudinal marker, Signorelli et al. (2021) and Devaux et al. (2023) proposed two methods, respectively called Penalized Regression Calibration (PRC) and DynForest, that can deal with a large number of longitudinal covariates. Notably, the estimation of PRC and DynForest is much more complex than that of LOCF landmarking, warranting dedicated software that can facilitate the implementation of such methods. PRC is implemented in the R package **pencal** (Signorelli, 2023), whereas DynForest in the R package **DynForest** (Devaux et al., 2023).

In this article we introduce the R package **pencal**, which implements the PRC approach. PRC uses mixed-effects models to describe and summarize the longitudinal biomarker trajectories; the summaries thus obtained are used as predictors of survival in a Cox model alongside with any relevant time-independent covariate. To account for the possible availability of a large (potentially high-dimensional) number of time-independent and longitudinal covariates and to reduce the risk of overfitting the training data, PRC uses penalized maximum likelihood to estimate the aforementioned Cox model.

The remainder of the article is organized as follow. In the next section we describe the statistical methodology behind **pencal**, including the dynamic prediction problem, PRC (Signorelli et al., 2021) and the problem of evaluating the model's predictive performance. Next we provide a general overview of the R package, discussing the implementation details of its functions for model estimation, prediction and performance validation. Furthermore, we provide a step by step example that shows how to use **pencal** to implement dynamic prediction on a real-world dataset that comprises several longitudinal covariates. We present the results of 4 simulations that assess the relationship between computing time and sample size, number of covariates and number of bootstrap samples used to validate model performance, showing how parallelization may reduce computing time significantly. Lastly, we provide some final remarks, and discuss limitations and possible extensions of the current approach.

Statistical methods

Input data and notation

We consider a setting where n subjects are followed from time $t = 0$ until an event of interest occurs. For each subject $i \in \{1, ..., n\}$ we observe the pair (t_i, δ_i), where δ_i is a dummy variable that indicates whether the event is observed at time $t = t_i$ ($\delta_i = 1$), or the observation of the event is right-censored at $t = t_i$ ($\delta_i = 0$). Thus, t_i corresponds to the survival time if $\delta_i = 1$, and to the censoring time if $\delta_i = 0$.

In addition to (t_i, δ_i), we assume that both baseline and follow-up information is collected from the same subjects, and that the number of variables gathered may be large. We consider a flexible unbalanced study design where the number and timing of the follow-up times can differ across subjects. We denote by $m_i \geq 1$ the number of repeated measurements available for subject i, and by $t_{i1}, ..., t_{im_i}$ $(t_{ij} \geq 0 \ \forall j)$ the corresponding follow-up times. For each subject i we observe:

1. a vector of k baseline (time-fixed) predictors $x_i = (x_{1i}, ..., x_{ki})$;
2. m_i vectors of p longitudinal (time-varying) predictors $y_{ij} = \left(y_{1ij}, ..., y_{pij}\right)$, $j = 1, ..., m_i$ measured at times $t_{i1}, ..., t_{im_i}$. Note that not all longitudinal predictors ought to be measured at every follow-up time t_{ij}, and the number of available measurements is thus allowed to differ across longitudinal predictors.

Dynamic prediction of survival

Let $S_i(t) = P(T_i > t)$ denote the survival function, i.e., the probability that subject i has not experienced the event up to time t, and let $S_i(t_B|t_A) = P(T_i > t_B|T_i > t_A)$, $t_B \geq t_A$ denote the conditional probability that subject i survives up until t_B, given that they survived up until t_A. Our goal is to predict the probability of survival of subject i given all the available information up until a given landmark time $t_L > 0$, namely:

$$S_i(t|t_L, x_i, \mathcal{Y}_i(t_L)) = P(T_i > t|T_i > t_L, x_i, \mathcal{Y}_i(t_L)), \tag{1}$$

where $\mathcal{Y}_i(t_L) = \{y_{i1}, ..., y_{ir} : t_{ir} \leq t_L\}$ denotes all repeated measurements available up to the landmark time for subject i.

In practice, often one may be interested in computing the predictions of survival in (1) over a range of q landmark times $t_{L1}, t_{L2}, ..., t_{Lq}$. By doing so, information from more and more repeated measurements can be incorporated in the prediction model as time passes, and predictions can be updated based on the latest available information. This approach is referred to as *dynamic prediction*, as it involves dynamic updates of model estimates and predictions over time. In this article we illustrate how to use **pencal** to compute predictions of the conditional survival probabilities in (1) for a single landmark time t_L. Note that implementing a dynamic prediction approach with **pencal** is straightforward, as for each landmark time t_{Ls} one simply needs to refit PRC over a dataset that comprises all repeated measurements available up until $t = t_{Ls}$ for the subjects that survived up until the landmark time t_{Ls} (i.e., including subject i if and only if $t_i \geq t_{Ls}$).

Penalized regression calibration

PRC (Signorelli et al., 2021) is a statistical method that makes it possible to estimate the conditional survival probabilities in (1) using x_i and $\mathcal{Y}_i(t_L)$ as inputs. The estimation of PRC requires a multi-step procedure that comprises the 3 following steps:

1. model the evolution over time of the longitudinal predictors in $\mathcal{Y}_i(t_L)$ using linear mixed models (LMM, McCulloch and Searle (2004)) or multivariate latent process mixed models (MLPMM, Proust-Lima et al. (2013));
2. use the model(s) fitted at step 1 to compute summaries of the trajectories described by the longitudinal predictors (i.e., the predicted random effects);
3. estimate a penalized Cox model for the survival outcome (t_i, δ_i) using as covariates both the baseline predictors x_i and the predicted random effects computed in step (2).

For simplicity, in this section we describe the version of PRC where in step 1 each longitudinal covariate is modelled using a separate LMM. This version of the model is referred to as PRC LMM in Signorelli et al. (2021). Note that alongside with the PRC LMM approach, Signorelli et al. (2021) also proposed a second approach called PRC MLPMM, where groups of longitudinal covariates are modelled jointly using the MLPMM. This alternative approach can be of interest when multiple longitudinal items are employed to measure the same underlying quantity (for example, multiple antibodies that target the same protein). For the formulation of the PRC MLPMM approach, we refer readers to Sections 2.1-2.3 of Signorelli et al. (2021) as the notation for steps 1 and 2 using the MLPMM is significantly more involved.

Denote by $\mathcal{I}(t_L) = \{i : t_i > t_L\}$ the set of subjects that survived up until the landmark time t_L. Let $y_{si} = (y_{si1}, ..., y_{sir})$, where $t_{ir} \leq t_L$ denotes the last follow-up time before t_L for subject i, be the vector that comprises all the measurements of the s-th longitudinal variable Y_s available up to the landmark time. In the first step of PRC, we model the evolution over time of each longitudinal covariate Y_s through a linear regression model

$$y_{si} = W_{si}\beta_s + Z_{si}u_{si} + \varepsilon_{si},\ i \in \mathcal{I}(t_L), \tag{2}$$

where β_s is a vector of fixed effect parameters, $u_{si} \sim N(0, D_s)$ is a vector of random effects, $\varepsilon_{si} \sim N(0, \sigma_s^2 I_{m_i})$ is the error term vector, and W_{si} and Z_{si} are design matrices associated to β_s and u_{si}. As an example, later in this article we will consider an example where we let y_{si} depend on the age a_{ij} of subject i at each visit, and include a random intercept and random slope in the LMM:

$$y_{sij} = \beta_{s0} + u_{si0} + \beta_{s1}a_{ij} + u_{si1}a_{ij} + \varepsilon_{ij}, \tag{3}$$

where $(u_{s0}, u_{s1})^T \sim N(0, D_s)$ is a vector of random effects that follows a bivariate normal distribution. We employ maximum likelihood (ML) estimation to estimate β_s, D_s and σ_s^2 in model (2).

In the second step of PRC, we use the ML estimates from step 1 to derive summaries of the individual longitudinal trajectories for each biomarker. These are the predicted random effects, which can be computed as

$$\hat{u}_{si} = E(u_{si}|Y_{si} = y_{si}) = \hat{D}_s Z_{si}^T \hat{V}_{si}^{-1}(y_{si} - X_{si}\hat{\beta}_s), \tag{4}$$

where $\hat{V}_{si} = Z_{si}\hat{D}_s Z_{si}^T + \hat{\sigma}_s^2 I$.

In the third step of PRC, we model the relationship between the survival outcome and the baseline and longitudinal predictors. This is achieved through the specification of a Cox model where we

include the baseline predictors x_i and the summaries of the longitudinal predictors $\hat{u}_i = (\hat{u}_{1i}, ..., \hat{u}_{pi})$ as covariates:

$$h(t_i|x_i, \hat{u}_i) = h_0(t_i) \exp(\gamma x_i + \delta \hat{u}_i), \tag{5}$$

where $h_0(t_i)$ is the baseline hazard function, and γ and δ are vectors of regression coefficients. Since our approach allows for the inclusion of a potentially large number of baseline and longitudinal covariates, we estimate model (5) using penalized maximum likelihood (PML, Verweij and Van Houwelingen (1994)). As penalties we consider the ridge (L_2), lasso (L_1) and elasticnet penalties. The elasticnet penalty for model (5) is given by

$$\lambda \left[\alpha \left(\sum_{s=1}^{p} |\gamma_s| + \sum_{j=1}^{p} |\delta_s| \right) + (1-\alpha) \left(\sum_{s=1}^{p} \gamma_s^2 + \sum_{s=1}^{p} \delta_s^2 \right) \right], \tag{6}$$

where $\lambda \geq 0$ and $\alpha \in [0, 1]$. The ridge penalty is obtained by setting $\alpha = 0$, and the lasso penalty by fixing $\alpha = 1$.

Computation of the predicted survival probabilities

Once models (2) and (5) have been estimated, the predicted survival probabilities $S_i(t|t_L, x_i, \mathcal{Y}_i(t_L))$ are computed using

$$\hat{S}_i(t|t_L, x_i, \mathcal{Y}_i(t_L)) = \exp\left(- \int_0^t \hat{h}_0(s) \exp(\hat{\gamma} x_i + \hat{\delta} \hat{u}_i) \right), \tag{7}$$

where $\hat{h}_0(s)$ is the estimated baseline hazard function, $\hat{\gamma}$ and $\hat{\delta}$ are the PML estimates of γ and δ obtained in step 3, and $\hat{u}_i$ contains the predicted random effects computed in step 2.

For subjects $i \in \mathcal{I}(t_L)$, who are included in the training set and survived up until t_L, computation of (7) is straightforward, since the predicted random effects $\hat{u}_i$ for such subjects have already been computed in step 2. Predictions of $S_i(t|t_L, x_i, \mathcal{Y}_i(t_L))$ for a new subject $i = n+1$ who survived up until t_L, but was not part of the training set is a bit more complex: before computing (7), one first needs to compute the predicted random effects for this new subject using (4). Note that such computation is feasible if and only if measurements of both baseline and longitudinal covariates (up to t_L) are available for this new subject.

Evaluation of the predictive performance

We consider the time-dependent area under the ROC curve (tdAUC, Heagerty et al. (2000)), the concordance index or C index (Pencina and D'Agostino, 2004) and the Brier score (Graf et al., 1999) as measures of predictive performance. To obtain unbiased estimates of these performance measures, Signorelli et al. (2021) proposed a cluster bootstrap optimism correction procedure (CBOCP) that generalizes the use of the bootstrap as internal validation method to problems involving repeated measurement data. As an alternative to the CBOCP, one may choose to implement a cross-validation approach instead. Should the user opt for such an alternative, we recommend the use of repeated cross-validation over simple cross-validation to achieve a level of accuracy comparable to that of the CBOCP.

The `R` package `pencal`

In this Section we introduce the functions for the estimation of PRC, the computation of the predicted survival probabilities and the validation of predictive performance, providing an overview of the relevant estimation approaches and some important implementation details.

Table 1 provides a side-by-side overview of the functions that can be used to implement the PRC LMM and PRC MLPMM approaches. Note that while two different functions (one for each approach) are needed for the three estimation steps and the computation of the survival probabilities, the evaluation of the predictive performance is implemented in a single function that works with inputs from both approaches.

Table 1: Overview of the pencal functions that implement the different modelling steps for the PRC LMM and PRC MLPMM approaches.

Task	PRC LMM	PRC MLPMM
Step 1: estimate the mixed-effects models	`fit_lmms`	`fit_mlpmms`
Step 2: compute the predicted random effects	`summarize_lmms`	`summarize_mlpmms`
Step 3: estimate the penalized Cox model	`fit_prclmm`	`fit_prcmlpmm`
Computation of predicted survival probabilities	`survpred_prclmm`	`survpred_prcmlpmm`
Evaluation of predictive performance	`performance_prc`	`performance_prc`

Model estimation and prediction

The first step of PRC involves the estimation of mixed-effects models for the longitudinal outcomes. For the PRC LMM approach, this can be done through the `fit_lmms` function, that proceeds to estimate p LMMs (one LMM for each of the longitudinal outcomes). Estimation of the LMMs is performed by maximum likelihood through the **lme** function from the `R` package **nlme** (Pinheiro and Bates, 2000) For the PRC MLPMM approach, the first step involves estimating one MLPMM for each group of longitudinal covariates. Estimation of the MLPMMs is done by maximum likelihood using the modified Marquardt algorithm described in Proust-Lima et al. (2013), as implemented in the `multlcmm` from the `R` package **lcmm** (Proust-Lima et al., 2017).

The second step of PRC requires the computation of the predicted random effects. The function `summarize_lmms` implements this for the PRC LMM approach. The function takes the output of `fit_lmms` as input, and proceeds to the computation of the predicted random effects using equation (4). Similarly, the function `summarize_mlpmms` does the same for the PRC MLPMM approach by taking the output of `fit_mlpmms` as input and computing the predicted random effects using the formula given in equation (4) of Signorelli et al. (2021).

The third step of PRC requires the estimation of a Cox model where the baseline covariates and the predicted random effects are used as covariates.Estimation of such model can be performed using the function `fit_prclmm` for the PRC LMM approach and `fit_prcmlpmm` for the PRC MLPMM approach. These functions proceed to the estimation of the aforementioned Cox model by penalized maximum likelihood through the function `cv.glmnet` from the `R` package **glmnet** (Simon et al., 2011). If the user chooses the ridge or lasso penalty, then the selection of the value of the tuning parameter λ is performed through cross-validation as implemented in **glmnet**. If, instead, the elasticnet penalty is used, `fit_prclmm` and `fit_prcmlpmm` proceed to perform a nested cross-validation procedure to jointly select the optimal values of the tuning parameters α and λ.

Lastly, the function `survpred_prclmm` can be used to compute the predicted survival probabilities as described in equation (7) for the PRC LMM approach. The corresponding function for the PRC MLPMM approach is `survpred_prcmlpmm`.

Computation of the CBOCP

In `pencal`, the evaluation of the predictive performance of the fitted model is done by estimating the tdAUC, C index and Brier score through the CBOCP described in Signorelli et al. (2021). For both the PRC LMM and PRC MLPMM approaches, this can be done through the function `performance_prc`. The estimates of the tDAUC, C index and Brier score are computed using functions from the packages **survivalROC** (Heagerty and Saha-Chaudhuri, 2022), **survcomp** (Schröder et al., 2011) and **riskRegression** (Gerds et al., 2023), respectively.

The computation of the CBOCP requires the choice of the number of bootstrap replicates B over which the model should be refitted. This can be done by specifying the argument `n.boots` inside the functions that implement the first step of PRC, namely `fit_lmms` for the PRC LMM approach and `fit_mlpmms` for the PRC MLPMM one. The supplied value of `n.boots` is stored in the output of such functions, and all subsequent functions inherit this value, automatically performing the computations necessary for the CBOCP.

If `n.boots = 0` (default), the CBOCP is not computed, and `performance_prc` only returns the naïve estimates of predictive performance. Values of `n.boots` ≥ 1 will trigger the computation of the CBOCP, and the output of `performance_prc` will additionally include the estimates of the optimism and the optimism-corrected performance measures. A typical value for B is 100, but in general we recommend setting B to a value between 50 and 200 (depending on computing time and the desired level of accuracy, one may also consider larger values of B).

User-friendly parallelization

The computation of the CBOCP is by nature repetitive, as it requires to repeat steps 1, 2 and 3 over B bootstrap samples, and to compute predictions and performance measures both on the bootstrap sample and on the original dataset to estimate the optimism. Due to this repetitiveness, such computations can be easily parallelized to reduce computing time. Moreover, also the estimation of the p LMMs / MLPMMs in step 1 can be trivially parallelized.

With the goal of making it as easy as possible for users to parallelize such computations, the functions in Table 1 automatically parallelize the aforementioned computations using the `%dopar%` operator from the `R` package **foreach** (Microsoft and Weston, 2022). The user only needs to specify the number of cores they want to use for the computation using the argument `n.cores`; **pencal** will automatically take care of the parallelization.

Classes, methods and further functionalities

Besides the core functions introduced in Table 1, **pencal** comprises additional functions that are shortly described hereafter.

Three functions are used to simulate the data that are used in the package documentation to illustrate typical usage of **pencal**'s functions. The function `simulate_t_weibull` is used to generate survival times from a Weibull distribution using the inverse transformation method. The functions `simulate_prclmm_data` and `simulate_prcmlpmm_data` are used to generate data for the estimation of PRC LMM and PRC MLPMM, respectively.

As the number of longitudinal covariates increases, step 1 of PRC will involve the estimation of many mixed-effects models. The functions `getlmm` and `getmlpmm` are provided to simplify the extraction of the estimates of each mixed model.

We employ S3 classes and methods for the final output of PRC, namely the penalized Cox model fitted in step 3 that is used to compute predictions of survival. `fit_prclmm` outputs an object of class `prclmm`, and `fit_prcmlpmm` one of class `prcmlpmm`. For both classes, `print` and `summary` methods are implemented.

Lastly, it may be sometimes of interest to compare the performance of PRC to that of either a penalized Cox model that only uses baseline values of all covariates, or a penalized Cox model with LOFC landmarking. The `pencox_baseline` function provides an interface to estimate these two models and to compute the associated CBOCP. Its output can be fed to the function `performance_pencox_baseline` to obtain the naïve and optimism-corrected estimates of the tdAUC, C index and Brier score for these two models.

Dynamic prediction with pencal: a step by step example

Loading `pbc2data`

To illustrate how to use **pencal** in practice, we employ data from a study from a clinical trial on primary biliary cholangitis (PBC) conducted by the Mayo Clinic from 1974 to 1984 (Murtaugh et al., 1994). The trial recorded the first of two survival outcomes, namely liver transplantation or death. In this example we focus our attention on the prediction of deaths, treating patients who underwent liver transplantation as right-censored. The data are available in **pencal** in a list called `pbc2data` that can be loaded as follows:

```
library(pencal)
data(pbc2data)
ls(pbc2data)
```

```
#> [1] "baselineInfo"     "longitudinalInfo"
```

`pbc2data` contains two data frames: `baselineInfo` records the survival information (t_i, δ_i) and baseline covariates x_i, whereas `longitudinalInfo` contains repeated measurements of the longitudinal predictors y_i. For simplicity, we rename the two data frames as `sdata` and `ldata`:

```
sdata = pbc2data$baselineInfo
ldata = pbc2data$longitudinalInfo
```

Input data format

As detailed in the Statistical Methods section, estimation of PRC requires the following input data for each subject $i = 1, ..., n$:

- a pair (t_i, δ_i) providing the survival outcome for subject $i = 1$;
- a vector of k baseline covariates x_i;
- an $m_i \times p$ matrix containing all repeated measurements of the p longitudinal predictors. Within this matrix each column corresponds to a predictor, and each row to the measurements y_{ij} collected at time t_{ij}, $j \in \{1, ..., m_i\}$ for subject i;
- any longitudinal covariate needed to construct the design matrices W_{si} and Z_{si} that will be used to estimate the LMMs of equation (2).

Such data should be provided to **pencal** using two data frames. The first data frame is a data frame that contains information on the survival outcomes (t_i, δ_i) and the baseline covariates x_i. For the PBC2 example, this is the `sdata` data frame created above:

```
head(sdata)
```

```
#>    id      time event baselineAge    sex treatment
#> 1   1  1.095170     1    58.76684 female D-penicil
#> 3   2 14.152338     0    56.44782 female D-penicil
#> 12  3  2.770781     1    70.07447   male D-penicil
#> 16  4  5.270507     1    54.74209 female D-penicil
#> 23  5  4.120578     0    38.10645 female   placebo
#> 29  6  6.853028     1    66.26054 female   placebo
```

This data frame should comprise at least 3 variables: a variable named `id` that contains subject identifiers, a variable named `time` containing the values of t_i, and a dummy variable named `event` that corresponds to δ_i (`event = 1` for subjects who experience the event at t_i, and event = 0 for right-censored observations). Additionally, it can also contain the baseline covariates x_i (if any); in the example we have $k = 3$ baseline covariates: `baselineAge`, `sex` and `treatment`.

The second data frame is a dataset in long format that contains the repeated measurements of the longitudinal predictors y_{ij} and of any covariate needed to create the design matrices W_{si} and Z_{si}. In our example application, such information is stored in `ldata`:

```
head(ldata)
```

```
#>   id      age   fuptime serBilir serChol albumin alkaline  SGOT platelets
#> 1  1 58.76684 0.0000000     14.5     261    2.60     1718 138.0       190
#> 2  1 59.29252 0.5256817     21.3      NA    2.94     1612   6.2       183
#> 3  2 56.44782 0.0000000      1.1     302    4.14     7395 113.5       221
#> 4  2 56.94612 0.4983025      0.8      NA    3.60     2107 139.5       188
#> 5  2 57.44716 0.9993429      1.0      NA    3.55     1711 144.2       161
#> 6  2 58.55054 2.1027270      1.9      NA    3.92     1365 144.2       122
#>   prothrombin
#> 1        12.2
#> 2        11.2
#> 3        10.6
#> 4        11.0
#> 5        11.6
#> 6        10.6
```

This "longitudinal" data frame should contain the following information:

- a variable named `id` that contains subject identifiers;
- a variable containing the time from baseline t_{ij} at which the measurement was collected. In `ldata`, we called this variable `fuptime` (short for: follow-up time);
- the p longitudinal predictors (`serBilir`, `serChol`, `albumin`, `alkaline`, `SGOT`, `platelets` and `prothrombin` in the example);
- the covariates needed to construct W_{si} and Z_{si} (age in the example).

Choice of the landmark time and data preparation

Before proceeding with the estimation of PRC and the computation of the predicted survival probabilities $S_i(t|t_L, x_i, \mathcal{Y}_i(t_L))$, three preliminary steps are needed. The first involves the choice of the landmark time t_L. Hereafter we choose $t_L = 2$ years, meaning that we want to predict the survival probability $S(t|t_L = 2, x_i, \mathcal{Y}_i(2))$ for a subject with baseline covariates x_i and longitudinal covariates measured up until two years from baseline $\mathcal{Y}_i(2)$.

```
# set the landmark time
lmark = 2
```

Once t_L has been chosen, the second step is to retain for analysis only those subjects that survived up until the landmark time, i.e. all $i : t_i \geq t_L$:

```
# remove subjects who had event / were censored before landmark
sdata = subset(sdata, time > lmark)
ldata = subset(ldata, id %in% sdata$id)
```

Lastly, only repeated measurements taken up to the landmark time ($t_{ij} \leq t_L$) should be retained for modelling, whereas measurements taken after the landmark time ($t_{ij} > t_L$) should be discarded:

```
# remove measurements taken after landmark:
ldata = subset(ldata, fuptime <= lmark)
```

Descriptive statistics, data visualization and transformation

After choosing $t_L = 2$ as landmark time, the number of subjects retained for model estimation is 278, of which 107 experience the event of interest, whereas the remaining 171 are right-censored:

```
# number of subjects retained in the analysis:
nrow(sdata)
```

```
#> [1] 278
```

```
# number of events (1s) and censored observations (0s):
table(sdata$event)
```

```
#>
#>   0   1
#> 171 107
```

The estimated survival probability can be visualized through the Kaplan-Meier estimator in Figure 1 as shown below:

```
library(survival)
library(survminer)
surv.obj = Surv(time = sdata$time, event = sdata$event)
KM = survfit(surv.obj ~ 1, type = "kaplan-meier")
ggsurvplot(KM, data = sdata, risk.table = T, cumevents = T, legend = 'none')
```

Spaghetti plots displaying the trajectory described by a longitudinal covariate, as well as density plots to visualize its marginal distribution, can be created in `ggplot2` style using:

```
library(ggplot2)
library(gridExtra)
traj1 = ggplot(ldata, aes(x = age, y = serBilir, group = id)) +
  geom_line(color = 'darkgreen') + theme_classic() + ggtitle('Trajectories of serBilir')
dens1 = ggplot(ldata, aes(x = serBilir)) +
  geom_density(adjust=1.5, alpha=.4, fill = 'orange') +
  theme_classic() + ggtitle('Density of serBilir')
grid.arrange(traj1, dens1, ncol = 2)
```

The two charts thus created are shown in Figure 2.

We can observe that some longitudinal covariates exhibit strong skewness, as in the case of `serBilir`. Although in principle the LMM can be used to model variables with skewed distributions, this may sometimes lead to converge problems or poor model fit. It can thus be advisable to transform such covariates to prevent these problems (but note that this is not a required modelling choice, and one may alternatively choose to avoid such transformation). For this reason, we log-transform those longitudinal covariates with skewed distribution before modelling them with LMMs:

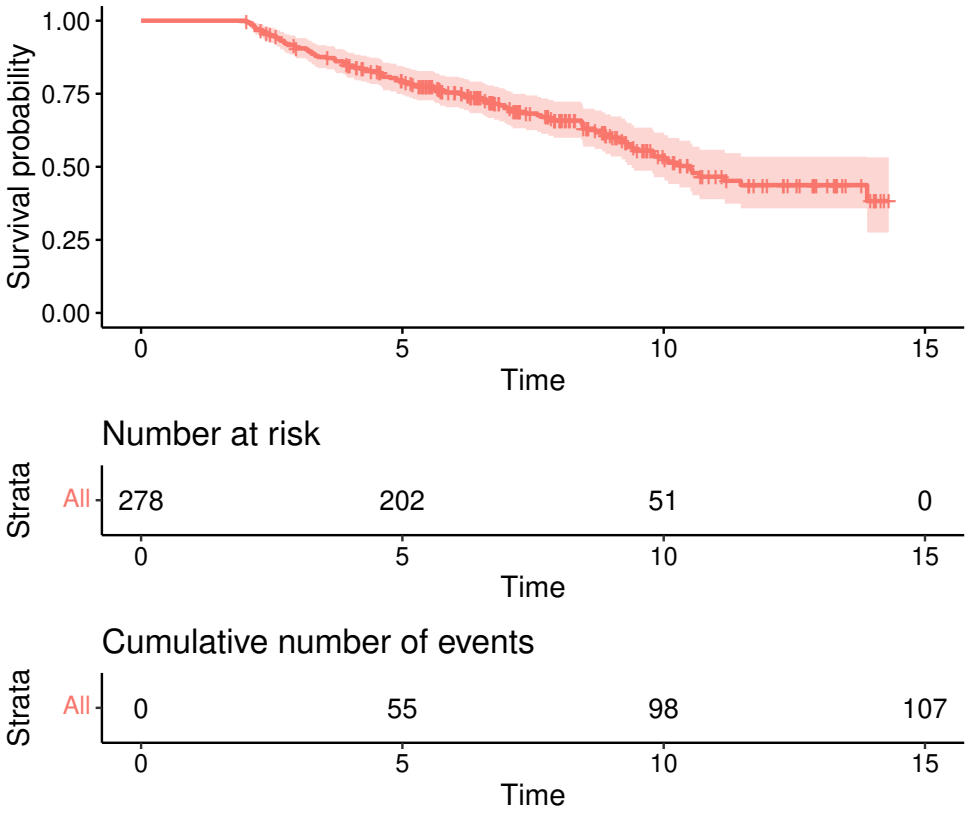

Figure 1: Chart displaying the Kaplan-Meier estimates of the conditional survival probability $S(t|2)$.

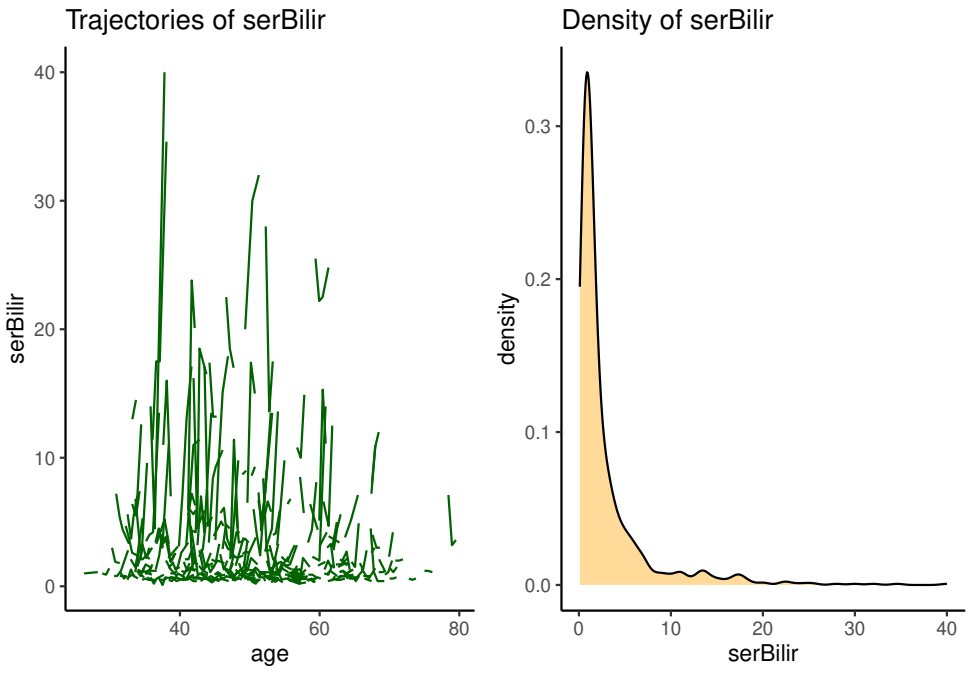

Figure 2: Spaghetti and density charts for the variable serBilir.

```
ldata$logSerBilir = log(ldata$serBilir)
ldata$logSerChol = log(ldata$serChol)
ldata$logAlkaline = log(ldata$alkaline)
ldata$logSGOT = log(ldata$SGOT)
ldata$logProthrombin = log(ldata$prothrombin)
```

Model estimation

Step 1: estimation of the mixed-effects models

The first step in the estimation of PRC involves modelling the evolution over time of the longitudinal predictors through mixed-effects models. Hereafter we model each of the longitudinal predictors using the LMM given in equation (3), which comprises a random intercept and a random slope for age. Estimation of this model for each longitudinal predictor can be done using the function `fit_lmms`:

```
lmms = fit_lmms(y.names = c('logSerBilir', 'logSerChol', 'albumin', 'logAlkaline',
                            'logSGOT', 'platelets', 'logProthrombin'),
                fixefs = ~ age, ranefs = ~ age | id,
                long.data = ldata, surv.data = sdata,
                t.from.base = fuptime, n.boots = 0,
                n.cores = 8, verbose = F)
```

The argument `y.names` is a character vector used to specify the names of the longitudinal predictors in `ldata`. `fixefs` and `ranefs` are formulas used to specify the fixed and random effects part of the LMM using the `nlme` formula notation (Pinheiro et al., 2022; Gałecki and Burzykowski, 2013). In the example, `fixefs = ~ age` determines the inclusion of the fixed effects part $\beta_{s0} + \beta_{s1}a_{ij}$ of model (3), and `ranefs = ~ age | id` the inclusion of the random effects part $u_{si0} + u_{si1}a_{ij}$ (allowing the random intercept and random slopes to be correlated).

The arguments `long.data` and `surv.data` are used to provide the names of the data frames containing the longitudinal variables (`long.data`) and the survival data and baseline covariates (`surv.data`) in the data formats described previously. The argument `t.from.base` is used to specify the name of the variable in `long.data` that contains the values of time from baseline; this argument is used internally to check that the data have been landmarked properly.

The `n.boots` argument is used to specify the number of bootstrap samples to use for the CBOCP. For the time being we focus on model estimation and on the prediction of the conditional probabilities, setting `n.boots = 0`; later we will show how to compute the CBOCP by setting `n.boots = 50`. Lastly, `n.cores` allows to specify the number of cores to use to parallelize computations (the default is `n.cores = 1`, i.e. no parallelization), and `verbose` is a logical value that indicates whether information messages should be printed in the console (`TRUE`, default) or not (`FALSE`). Additional arguments are described in the help page, see `?fit_lmms`.

The function `getlmm` can be used to obtain the parameter estimates from the mixed models. For example, to obtain the parameters of the LMM for `logSerBilir` we can use:

```
# get estimates from fitted LMMs:
getlmm(lmms, yname = 'logSerBilir', what = 'betas')
```

```
#>  (Intercept)          age
#>  0.518319707 -0.001044517
```

```
getlmm(lmms, yname = 'logSerBilir', what = 'variances')
```

```
#> id = pdLogChol(age)
#>             Variance     StdDev      Corr
#> (Intercept) 7.332118e-01 0.856277849 (Intr)
#> age         4.731627e-05 0.006878682 0.103
#> Residual    1.437622e-01 0.379159888
```

From the output we can deduce that the ML estimates for the LMM involving `logSerBilir` are $\hat{\beta} = (-4.845, 0.108)$, $\hat{\sigma}^2_{u0} = 33.63$, $\hat{\sigma}^2_{u1} = 0.013$, and $\hat{\sigma}_{u0,u1} = -0.953 \cdot \sqrt{33.63 \cdot 0.013}$. The usual table with parameter estimates, standard error and p-values can be obtained with

```
getlmm(lmms, yname = 'logSerBilir', what = 'tTable')
```

```
#>                    Value   Std.Error  DF    t-value    p-value
#> (Intercept)  0.518319707 0.278822874 566  1.8589569 0.06355204
#> age         -0.001044517 0.005543207 566 -0.1884318 0.85060568
```

Step 2: computation of the predicted random effects

After the ML estimates from the LMM have been computed, the second step of PRC involves the computation of the predicted random effects $\hat{u}_{si}$. Such computation can be performed using the function `summarize_lmms`:

```
pred_ranefs = summarize_lmms(object = lmms, n.cores = 8, verbose = F)
```

Here, the `object` argument is used to pass the output of `fit_lmms` to `summarize_lmms`; `n.cores` and `verbose` are the same as in `fit_lmms`. The output of `summarize_lmms` is a list that contains, among other elements, a matrix with the predicted random effects called `ranef.orig`:

```
round(pred_ranefs$ranef.orig[1:4, 1:4], 4)
```

```
#>   logSerBilir_b_int logSerBilir_b_age logSerChol_b_int logSerChol_b_age
#> 2           -0.3830           -0.0017          -0.0712           0.0007
#> 3           -0.1171           -0.0006          -0.5985           0.0049
#> 4            0.1686            0.0009          -0.3704           0.0035
#> 5            0.3800            0.0012          -0.2910           0.0029
```

From the output we can deduce, for example, that the predicted random intercept and random slope for `logSerBilir` and subject 4 are $\hat{u}_{1,0,4} = 0.1686$ and $\hat{u}_{1,1,4} = 0.0009$.

Step 3: estimation of the penalized Cox model

The last step in the estimation of PRC involves the estimation of model (5) through PML. This can be achieved with the `fit_prclmm` function:

```
pencox = fit_prclmm(object = pred_ranefs, surv.data = sdata,
                    baseline.covs = ~ baselineAge + sex + treatment,
                    penalty = 'ridge', standardize = T,
                    n.cores = 8, verbose = F)
```

The `object` argument is used to pass the output of `summarize_lmms` to `fit_prclmm`; `surv.data` is the data frame that contains the information about survival data and baseline covariates; `baseline.covs` is a formula used to define which baseline covariates x_i should be included in model (5) (with associated regression coefficient γ). The `penalty` argument is a character that can take one of the following values:

1. `penalty = 'ridge'` to estimate model (5) using the ridge or L2 penalty within the PML estimation;
2. `penalty = 'lasso'` to estimate model (5) using the lasso or L1 penalty;
3. `penalty = 'elnet'` to estimate model (5) using the elasticnet penalty, (Zou and Hastie, 2005). If the elasticnet penalty is chosen, additional arguments such as `n.alpha.elnet` and `n.folds.elnet` can be specified to determine how to select the additional tuning parameter (α) used by this penalty through nested cross-validation.

The `standardize` argument is used to determine whether the predicted random effects should be standardized prior to inclusion in the Cox model (default is `TRUE`). The `n.cores` and `verbose` arguments are the same as in `fit_lmms`. See `?fit_prclmm` for a description of further arguments.

The output of `fit_prclmm` is an object of class `prclmm`, for which `print` and `summary` methods are available:

```
print(pencox, digits = 4)
```

```
#> Fitted model: PRC-LMM
#> Penalty function used: ridge
#> Sample size: 278
#> Number of events: 107
#> Bootstrap optimism correction: not computed
#> Penalized likelihood estimates (rounded to 4 digits):
#>   baselineAge sexfemale treatmentD-penicil logSerBilir_b_int logSerBilir_b_age
#> 1      0.0508   -0.3079            0.0153           0.5724          144.7131
#>   logSerChol_b_int logSerChol_b_age albumin_b_int albumin_b_age
#> 1           0.0675          -7.7419      -1.5212      34010.07
#>   logAlkaline_b_int logAlkaline_b_age logSGOT_b_int logSGOT_b_age
#> 1            0.0851           -1.3713        0.1718      260.7198
#>   platelets_b_int platelets_b_age logProthrombin_b_int logProthrombin_b_age
#> 1         -0.0014         -0.0579               3.0299            -141.3234
```

```
step3summary = summary(pencox)
ls(step3summary)

#> [1] "coefficients" "data_info"    "model_info"
```

The PML estimates of γ and δ can be obtained through

```
step3summary$coefficients

#>   baselineAge  sexfemale treatmentD-penicil logSerBilir_b_int logSerBilir_b_age
#> 1  0.05084829 -0.3078532         0.01528107           0.57237          144.7131
#>   logSerChol_b_int logSerChol_b_age albumin_b_int albumin_b_age
#> 1       0.06754837        -7.741941     -1.521196      34010.07
#>   logAlkaline_b_int logAlkaline_b_age logSGOT_b_int logSGOT_b_age
#> 1        0.08508773         -1.371284     0.1717898      260.7198
#>   platelets_b_int platelets_b_age logProthrombin_b_int logProthrombin_b_age
#> 1    -0.001370013     -0.05792844             3.029942            -141.3234
```

Computing predictions

The function `survpred_prclmm` can be used to compute the conditional survival probabilities $\hat{S}_i(t|t_L, x_i, \mathcal{Y}_i(t_L))$, $t \geq t_L$ in (7). For the subjects that have been used to estimate PRC, such computation can be performed using `survpred_prclmm` as follows:

```
preds = survpred_prclmm(step1 = lmms, step2 = pred_ranefs,
                        step3 = pencox, times = 3:7)
```

The `step1`, `step2` and `step3` arguments are used to pass the outputs of the 3 estimation steps to the function; the `times` argument is a vector with the prediction times at which one wishes to evaluate the conditional survival probabilities. The predicted survival probabilities are stored in the `predicted_survival` element of the function output:

```
ls(preds)

#> [1] "call"               "predicted_survival"

head(preds$predicted_survival)

#>   id      S(3)      S(4)      S(5)      S(6)      S(7)
#> 2  2 0.9590270 0.9190494 0.8760220 0.8321080 0.7590547
#> 3  3 0.8661607 0.7483216 0.6347010 0.5319358 0.3879723
#> 4  4 0.8224818 0.6741316 0.5388527 0.4237748 0.2758791
#> 5  5 0.9443926 0.8909712 0.8344221 0.7777516 0.6859084
#> 6  6 0.9501909 0.9020436 0.8507390 0.7989494 0.7141397
#> 7  7 0.9799139 0.9598852 0.9378204 0.9147176 0.8748472
```

Prediction for new subjects that have not been used for model estimation is a bit more involved, as it additionally requires to compute the predicted random effects for the new subjects using equation (4) based on the parameter estimates obtained in the first step of PRC. For illustration purposes, suppose that we have data from 3 subjects stored in two data frames called `new_ldata` and `new_sdata`:

```
new_ldata = subset(ldata, id %in% c(5, 54, 110))
new_sdata = subset(sdata, id %in% c(5, 54, 110), c('id', 'baselineAge', 'sex', 'treatment'))
head(new_ldata)

#>     id      age   fuptime serBilir serChol albumin alkaline  SGOT platelets
#> 23   5 38.10645 0.0000000      3.4     279    3.53      671 113.2       136
#> 24   5 38.65130 0.5448472      1.9      NA    3.28      689 103.9       114
#> 25   5 39.17698 1.0705290      2.5      NA    3.34      652 117.8        99
#> 419 54 39.19888 0.0000000      1.3     288    3.40     5487  73.5       254
#> 420 54 39.69171 0.4928266      1.5      NA    3.22     1580  71.3       112
#> 421 54 40.19001 0.9911291      2.6      NA    3.48     2127  86.8       207
#>     prothrombin logSerBilir logSerChol logAlkaline  logSGOT logProthrombin
#> 23         10.9   1.2237754   5.631212    6.508769 4.729156       2.388763
#> 24         10.7   0.6418539         NA    6.535241 4.643429       2.370244
#> 25         10.5   0.9162907         NA    6.480045 4.768988       2.351375
```

```
#> 419        11.0   0.2623643   5.662960    8.610137 4.297285       2.397895
#> 420        18.0   0.4054651         NA    7.365180 4.266896       2.890372
#> 421        10.9   0.9555114         NA    7.662468 4.463607       2.388763
```

```
head(new_sdata)
```

```
#>      id baselineAge    sex treatment
#> 23    5    38.10645 female   placebo
#> 419  54    39.19888 female D-penicil
#> 859 110    38.91140 female D-penicil
```

Note that the variables and their variable type in `new_ldata` and `new_sdata` should be the same as in the `ldata` and `sdata`; the only exception to this is that `new_sdata` does not need to contain information about survival, so unlike `sdata` it does not comprise the `time` and `event` variables.

To compute predicted probabilities for new subjects, it is once again possible to resort to `survpred_prclmm`; now, it is necessary to specify the arguments `new.longdata` and `new.basecovs` to supply the data about the new subjects to `survpred_prclmm`:

```
pred_new = survpred_prclmm(step1 = lmms, step2 = pred_ranefs,
                           step3 = pencox, times = 3:7,
                           new.longdata = new_ldata,
                           new.basecovs = new_sdata)
pred_new$predicted_survival
```

```
#>      id      S(3)      S(4)      S(5)      S(6)      S(7)
#> 5     5 0.9443926 0.8909712 0.8344221 0.7777516 0.6859084
#> 54   54 0.9230954 0.8508944 0.7763265 0.7035940 0.5901872
#> 110 110 0.9569132 0.9149666 0.8699275 0.8240807 0.7480976
```

Evaluation of the predictive performance

As explained in the Statistical Methods Section, estimation of the predictive performance in **pencal** is done through a CBOCP that allows to obtain unbiased estimates of predictive performance as measured by the tdAUC, C index and Brier score.

Before computing the (potentially time-consuming) CBOCP, one may want to first have a look at the naïve (biased) estimates of predictive performance. This can be done using the function `performance_prc`:

```
# naive performance (biased, optimistic estimate)
naive_perf = performance_prc(step2 = pred_ranefs, step3 = pencox,
                             metric = 'tdauc', times = 3:7,
                             n.cores = 8, verbose = F)
```

```
#> Warning in performance_prc(step2 = pred_ranefs, step3 = pencox, metric =
#> "tdauc", : The cluster bootstrap optimism correction has not been performed
#> (n.boots = 0). Therefore, only the apparent values of the performance values
#> will be returned.
```

Here `pred_ranefs` is the output of step2 of PRC and `pencox` the output of step 3. The `metric` argument can be used to specify the performance measures to be computed (possible values are `tdauc`, `c` and `brier`), whereas the `times` argument is used to specify the time points at which the tdAUC should be evaluated ($t = 3, 4, 5, 6, 7$ in this example). Notice that when `fit_lmms` has been run with `n.boots = 0`, `performance_prc` returns a warning to inform users that the CBOCP has not been performed; for now, we can ignore this warning (which we will address soon by refitting PRC with `n.boots = 50`).

```
naive_perf
```

```
#> $call
#> performance_prc(step2 = pred_ranefs, step3 = pencox, metric = "tdauc",
#>     times = 3:7, n.cores = 8, verbose = F)
#>
#> $tdAUC
#>   pred.time tdAUC.naive optimism.correction tdAUC.adjusted
#> 1         3      0.9434                 NA             NA
```

```
#> 2         4      0.9361                NA            NA
#> 3         5      0.9298                NA            NA
#> 4         6      0.9015                NA            NA
#> 5         7      0.8879                NA            NA
```

From the output we can observe that the naïve estimate of the tdAUC ranges from 0.9434 for predictions of survival at $t = 3$ up to 0.8879 for predictions at $t = 7$. These naïve (in-sample) measurements of predictive performance may be optimistically biased due to overfitting, i.e., the fact that they are evaluated using the same data on which PRC was estimated. To correct for this potential source of bias, below we show how to implement the CBOCP to obtain unbiased estimates of the tdAUC and C index.

Computation of the CBOCP requires to repeat the 3 estimation steps of PRC for each bootstrap samples; this can be done by rerunning the functions `fit_lmms`, `summarize_lmms` and `fit_prclmm` with the same arguments used previously, but setting `n.boots` within `fit_lmms` to an integer value larger than 0. `n.boots` specifies the number of bootstrap samples to use to compute the CBOCP. In the example below we set `n.boots = 50` (note that larger values of `n.boots` can increase the accuracy of the CBOCP estimates, but at the same time they increase computing time).

```
step1 = fit_lmms(y.names = c('logSerBilir', 'logSerChol', 'albumin', 'logAlkaline',
                             'logSGOT', 'platelets', 'logProthrombin'),
                 fixefs = ~ age, ranefs = ~ age | id,
                 long.data = ldata, surv.data = sdata,
                 t.from.base = fuptime, n.boots = 50,
                 n.cores = 8, verbose = F)
step2 = summarize_lmms(object = step1, n.cores = 8, verbose = F)
step3 = fit_prclmm(object = step2, surv.data = sdata,
                   baseline.covs = ~ baselineAge + sex + treatment,
                   penalty = 'ridge', n.cores = 8, verbose = F)
```

Once all computations are finished, it suffices to supply the refitted outputs of step 2 and step 3 to `performance_prc`:

```
# bootstrap-corrected performance (unbiased estimate)
cbocp = performance_prc(step2 = step2, step3 =  step3,
                        metric = c('tdauc', 'brier'), times = 3:7,
                        n.cores = 8, verbose = F)
cbocp
```

```
#> $call
#> performance_prc(step2 = step2, step3 = step3, metric = c("tdauc",
#>     "brier"), times = 3:7, n.cores = 8, verbose = F)
#>
#> $tdAUC
#>   pred.time tdAUC.naive optimism.correction tdAUC.adjusted
#> 1         3      0.9434             -0.0059         0.9375
#> 2         4      0.9348             -0.0149         0.9199
#> 3         5      0.9273             -0.0131         0.9142
#> 4         6      0.9002             -0.0090         0.8912
#> 5         7      0.8879             -0.0116         0.8763
#>
#> $Brier
#>   pred.time Brier.naive optimism.correction Brier.adjusted
#> 1         3      0.0556              0.0126         0.0682
#> 2         4      0.0679              0.0238         0.0917
#> 3         5      0.0823              0.0287         0.1110
#> 4         6      0.0929              0.0308         0.1237
#> 5         7      0.0985              0.0380         0.1365
```

In the outputs above, the column `tdAUC.naive` contains the naïve estimates of the tdAUC; `optimism.correction` reports the values of the estimated optimism correction from the CBOCP; finally, `tdAUC.adjusted` contains the unbiased estimates of the tdAUC.

As expected, the unbiased estimates of predictive performance are somewhat lower than the naïve ones. For example, the tdAUC estimate for predictions at $t = 3$ is 0.9375 instead of the naïve estimate 0.9434.

Evaluation of computing time

We now turn our attention to the relationship between the sample size n, number of longitudinal covariates p and number of bootstrap replicates B on computing time. Furthermore, we look into how parallel computing may be used to reduce computing time for the CBOCP. To gain insight into these relationships, we simulate data from the PRC LMM model using the function `simulate_prclmm_data` according to four simulation scenarios:

- in simulation 1 we study the effect of n on the estimation of PRC. To this aim, we let $n \in \{100, 200, 400, 600, 800, 1000\}$ and fix $p = 10$;
- in simulation 2 we study the effect of p on the estimation of PRC by taking $p \in \{5, 10, 20, 30, 40, 50\}$ and fixing $n = 200$;
- in simulation 3 we shift our attention to the effect of B on the computing time of the CBOCP. We let $B \in \{50, 100, 200, 300, 400, 500\}$, fixing $n = 200$ and $p = 10$;
- finally, in simulation 4 we compute PRC and the CBOCP on a dataset where $n = 200$, $p = 50$ and $B = 50$ using an increasing number of cores, namely $\{1, 2, 3, 4, 8, 16\}$.

Computations were performed on an AMD EPYC 7662 processor with 2 GHz CPU, using a single core for simulations 1, 2 and 3, and a number of cores ranging from 1 to 16 in simulation 4. Computing time was measured using the `rbenchmark` package (Kusnierczyk, 2012).

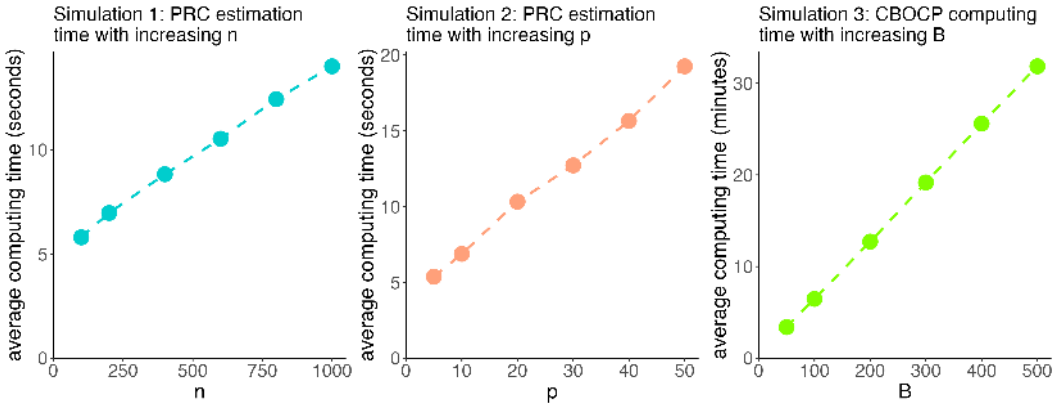

Figure 3: Left and center: average computing time (in seconds) of the estimation of the PRC LMM model as a function of the sample size n (simulation 1, left) and of the number of longitudinal predictors p (simulation 2, center). Right: average computing time (in minutes) for the computation of the CBOCP as a function of B (simulation 3).

The charts in Figure 3 display the average computing time over 10 replications of simulations 1, 2 and 3. In simulation 1, the average computing time increases from 5.80 seconds when $n = 100$ to 14.01 seconds when $n = 1000$, so we observe a 2.4-fold increase in computing time as n increases by a factor of 10. In simulation 2, the average computing time increases from 5.37 seconds when $p = 5$ to 19.26 seconds when $p = 50$, yielding a 3.6-fold increase in computing time as p increases by a factor of 10. Lastly, in simulation 3 the average computing time increases from 3.38 minutes when $B = 50$ to 32.32 minutes when $B = 500$, with a 9.6-fold time increase corresponding to a 10-fold increase in B.

Overall, the results of simulations 1 and 2 indicate that computing time for the estimation of PRC increases linearly with, but less than proportionally to, n and p (meaning that a k-fold increase in n or p will typically increase computing time by a factor smaller than k). Instead, from simulation 3 we can observe that the computing time of the CBOCP increases proportionally to B.

The results of simulations 1, 2 and 3 show that whereas the estimation of the PRC model itself typically requires only a few seconds, the computation of the CBOCP is more intensive and can require several minutes. This is due to the fact that the CBOCP requires the PRC modelling steps to be repeated on each bootstrap sample, effectively requiring to compute PRC $B + 1$ times (once on the original dataset + B times on the B bootstrap datasets). To reduce the computing time needed to compute the CBOCP, `pencal` enables users to easily parallelize computations through the argument `n.cores` within `fit_lmms`, `summarize_lmms` and `fit_prclmm`. In simulation 4, we show the effect that increasing the number of cores has on the computing time of the CBOCP.

Figure 4 shows the average computing time of the CBOCP over 10 replications of simulation 4. When looking at the total computing time, we can see that increasing the number of cores from 1 to 8 progressively decreases computing time, reducing it from 863.8 seconds without parallelization up to 235.7 seconds when using 8 cores (-68%). The most significant time gains are from 1 to 2 cores (-407.2

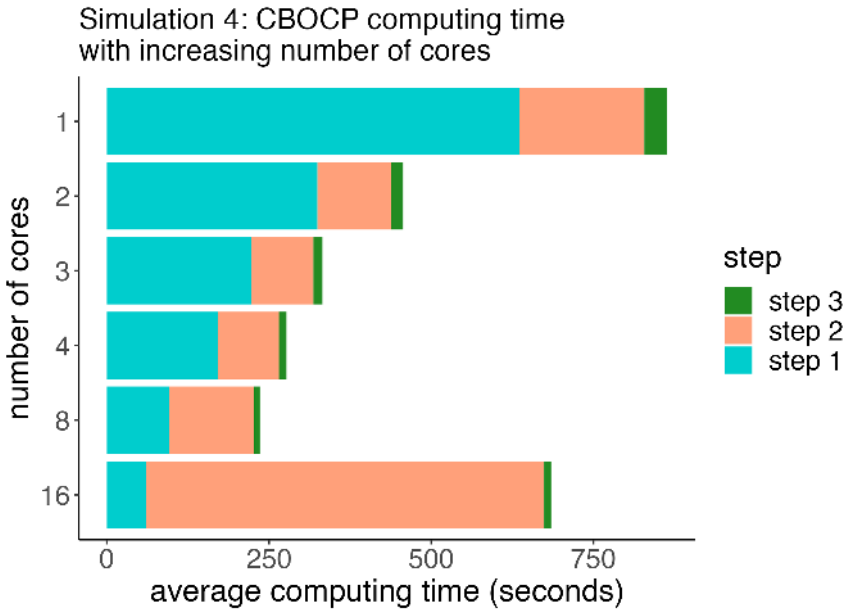

Figure 4: Average computing time (in seconds) for the estimation of the PRC LMM model and the computation of the CBOCP as a function of the number of cores.

seconds) and then from 2 to 3 (-124.4 seconds). Interestingly, further doubling the number of cores from 8 to 16 proves to be detrimental, increasing computing time from 235.7 to 684.5 seconds.

To understand this initially decreasing, but later increasing pattern, it is useful to consider the computing time of each of the modelling steps separately. By looking at Figure 4 we notice that step 1, which involves the estimation of $p \cdot (B + 1)$ LMMs, is the most time-consuming step; its computing time consistently decreases as the number of cores decreases (from 636.1 to 60.8 seconds).

The same does not apply to step 2, where computing time decreases from 1 to 4 cores (from 192.3 to 93.6 seconds), but then increases considerably when going from 8 (130.4 seconds) to 16 cores (613.2 seconds). This pattern is primarily due to the fact that step 2 mostly involves simple linear algebra: parallelizing this step on a large number of cores may be detrimental, as the (limited) time gain that can be achieved by doing these simple computations in parallel may be more than compensated by the time cost of dispatching the necessary matrices and vectors to many cores and recombining the results at the end of the parallelization.

As concerns step 3, we can see that it is the lightest step in terms of computing time. The pattern is consistently decreasing from 1 (35.4 seconds) to 8 cores (8.8 seconds), with a slight increase when using 16 cores (10.6 seconds).

In conclusion, the results of simulation 4 show how it may be advisable to parallelize computations to compute the CBOCP, but without using an excessive number of cores (specially for step 2). Our general advice is to use between 3 and 8 cores for optimal performance, nevertheless we emphasize how the effect of the number of cores on computing time may differ from the patterns in Figure 4 depending on a combination of factors such as n, p, number of repeated measurements per subject, and B. Furthermore, we notice that within `pencal`, a different number of cores can be chosen for each modelling step; in the example of simulation 4, the optimal performance would be achieved using 16 cores for step 1, 4 cores for step 2, and 8 cores for step 3 (but using 8 cores for all 3 steps isn't much less efficient).

Summary and discussion

The `R` package **pencal** provides a user-friendly implementation of Penalized Regression Calibration (PRC, Signorelli et al. (2021)), a statistical method that can be used to implement dynamic prediction of time-to-event outcomes in longitudinal studies where both time-independent and longitudinal (i.e., time-dependent) covariates are available as possible predictors of survival. The package comprises not only functions for the estimation of PRC and the prediction of survival, but also functions to compute unbiased estimates of predictive performance through a cluster bootstrap procedure. Because computing such bootstrap procedure may be time-consuming, the package automatically parallelizes repetitive computations using the `%dopar%` operator from the **foreach** package (Microsoft and Weston,

2022).

pencal focuses on problems where a single survival outcome is measured with right-censoring. As such, it is not designed to handle interval censoring or competing risks. The modelling of the longitudinal covariates is performed using either the LMM or the MLPMM, which are linear models that are mostly suitable for the analysis of continuous outcomes. Implementing generalized linear mixed models (GLMMs) would make it possible to properly deal with binary and discrete longitudinal covariates, however the estimation of GLMMs and the computation of the predicted random effects are more time-consuming and more prone to convergence problems, two aspects that would particularly complicate the computation of the CBOCP. For this reason we did not pursue GLMMs further, but leave them as a topic of future research. Users dealing with discrete longitudinal covariates may consider log-transforming them before modelling with a LMM within **pencal** (specially if such covariates are right-skewed and/or exhibit overdispersion).

Two modelling choices deserve particular attention when implementing PRC in specific application contexts. The first refers to the choice of the covariates to include in the fixed and random effects parts of the LMM of Equation (2). In principle, one may want to model the response variable as flexibly as possible, including several fixed effect covariates and multiple random effects in the LMM. However, when doing this one should consider that the purpose of the LMM is to provide subject-specific summaries of the individual trajectories. Thus, *the primary goal of the LMM in step 1 is that of obtaining predicted random effects that are good summaries of how the trajectory of a given subject differs from the population average*. In practice, such purpose may be more easily achieved using a simple mixed model that allows for a clear interpretation of its random effects, rather than using a complex one where the interpretation of each random effect may be unclear / complicated. The LMM of equation (3) (or, alternatively, the same model with follow-up time included as covariate instead of age) is a good example of simple LMM with clearly interpretable random effects, as the random intercept allows to distinguish subjects with high and low initial levels of the covariate, and the random slope to identify subjects with faster and slower progression rates. Therefore, even though `fit_lmms` makes it possible to consider complex fixed and random effects formulas, we still advise users to consider simpler mixed models in step 1 (and to compare the predictive performance of PRC using either approach, eventually choosing the approach that delivers more accurate predictions if there is a substantial difference).

A second important modelling choice when using **pencal** is which penalty function should be used in step 3. In general, this is a modelling choice that may dependent on the specific application and its features (sample size, number of predictors and number of available repeated measurements per subject). Signorelli et al. (2021) performed several simulation studies focused on situations with small sample sizes ($n = 100$ and $n = 300$) and sparse data generating processes for the survival outcome, whose results showed that the ridge and elasticnet penalty yielded better performance than the lasso penalty. Our experience is that in general the ridge penalty may be preferable both to elasticnet and the lasso in scenarios with small or moderate sample sizes, where little information is available to estimate the α tuning parameter of elasticnet or to reliably perform variable selection with the lasso. Beyond this, it is always possible to use a data-driven approach to choose which penalty to use by estimating PRC using the 3 different penalties and comparing how this affects predictive performance.

Software and code availability

The `R` package **pencal** can be downloaded from CRAN at cran.r-project.org/package=pencal. The development version of the package is available on Github at github.com/mirkosignorelli/pencal_devel. The code used in the simulations for the evaluation of computing time is available at github.com/mirkosignorelli/pencal_sims/.

Acknowledgements

The author kindly acknowledges funding from the Netherlands eScience Center Fellowship Programme.

Bibliography

A. Devaux, C. Proust-Lima, and R. Genuer. Random Forests for time-fixed and time-dependent predictors: The DynForest R package. *arXiv preprint arXiv:2302.02670*, 2023. [p2]

A. Gałecki and T. Burzykowski. Linear mixed-effects model. In *Linear Mixed-Effects Models Using R*, pages 245–273. Springer, 2013. [p10]

T. A. Gerds, J. S. Ohlendorff, and B. Ozenne. riskRegression: Risk Regression Models and Prediction Scores for Survival Analysis with Competing Risks, 2023. URL https://CRAN.R-project.org/package=riskRegression. R package version 2023.03.22. [p5]

E. Graf, C. Schmoor, W. Sauerbrei, and M. Schumacher. Assessment and comparison of prognostic classification schemes for survival data. *Statistics in Medicine*, 18(17-18):2529–2545, 1999. [p4]

P. J. Heagerty and P. Saha-Chaudhuri. survivalROC: Time-Dependent ROC Curve Estimation from Censored Survival Data, 2022. URL https://CRAN.R-project.org/package=survivalROC. R package version 1.0.3.1. [p5]

P. J. Heagerty, T. Lumley, and M. S. Pepe. Time-dependent ROC curves for censored survival data and a diagnostic marker. *Biometrics*, 56(2):337–344, 2000. [p4]

R. Henderson, P. Diggle, and A. Dobson. Joint modelling of longitudinal measurements and event time data. *Biostatistics*, 1(4):465–480, 2000. [p1]

G. L. Hickey, P. Philipson, A. Jorgensen, and R. Kolamunnage-Dona. joineRML: a joint model and software package for time-to-event and multivariate longitudinal outcomes. *BMC Medical Research Methodology*, 18:1–14, 2018. [p1]

W. Kusnierczyk. rbenchmark: Benchmarking routine for R, 2012. URL https://CRAN.R-project.org/package=rbenchmark. R package version 1.0.0. [p15]

C. E. McCulloch and S. R. Searle. *Generalized, Linear, and Mixed Models*. John Wiley & Sons, 2004. [p3]

Microsoft and S. Weston. *foreach: Provides Foreach Looping Construct*, 2022. URL https://CRAN.R-project.org/package=foreach. R package version 1.5.2. [p6, 16]

P. A. Murtaugh, E. R. Dickson, G. M. Van Dam, M. Malinchoc, P. M. Grambsch, A. L. Langworthy, and C. H. Gips. Primary biliary cirrhosis: prediction of short-term survival based on repeated patient visits. *Hepatology*, 20(1):126–134, 1994. [p6]

M. J. Pencina and R. B. D'Agostino. Overall C as a measure of discrimination in survival analysis: model specific population value and confidence interval estimation. *Statistics in Medicine*, 23(13): 2109–2123, 2004. [p4]

P. Philipson, I. Sousa, P. Diggle, P. Williamson, R. Kolamunnage-Dona, R. Henderson, and G. Hickey. joineR: Joint Modelling of Repeated Measurements and Time-to-Event Data, 2018. URL https://cran.r-project.org/package=joineR. R package version 1.2.8. [p1]

J. Pinheiro, D. Bates, and R Core Team. *nlme: Linear and Nonlinear Mixed Effects Models*, 2022. URL https://CRAN.R-project.org/package=nlme. R package version 3.1-160. [p10]

J. C. Pinheiro and D. M. Bates. *Mixed-Effects Models in S and S-PLUS*. Springer, 2000. [p5]

C. Proust-Lima, H. Amieva, and H. Jacqmin-Gadda. Analysis of multivariate mixed longitudinal data: a flexible latent process approach. *British Journal of Mathematical and Statistical Psychology*, 66(3): 470–487, 2013. [p3, 5]

C. Proust-Lima, V. Philipps, and B. Liquet. Estimation of extended mixed models using latent classes and latent processes: The R package cmm. *Journal of Statistical Software*, 78(2):1–56, 2017. [p5]

H. Putter and H. C. van Houwelingen. Landmarking 2.0: Bridging the gap between joint models and landmarking. *Statistics in Medicine*, 41(11):1901–1917, 2022. [p2]

D. Rizopoulos. JM: An R package for the joint modelling of longitudinal and time-to-event data. *Journal of Statistical Software*, 35:1–33, 2010. [p1]

D. Rizopoulos. The R package JMbayes for fitting joint models for longitudinal and time-to-event data using MCMC. *arXiv preprint arXiv:1404.7625*, 2014. [p1]

M. S. Schröder, A. C. Culhane, J. Quackenbush, and B. Haibe-Kains. survcomp: an R/Bioconductor package for performance assessment and comparison of survival models. *Bioinformatics*, 27(22): 3206–3208, 2011. [p5]

M. Signorelli. pencal: Penalized Regression Calibration (PRC) for the Dynamic Prediction of Survival, 2023. URL https://cran.r-project.org/package=pencal. R package version 2.1.0. [p2]

M. Signorelli, P. Spitali, C. Al-Khalili Sgyziarto, The Mark-MD Consortium, and R. Tsonaka. Penalized regression calibration: a method for the prediction of survival outcomes using complex longitudinal and high-dimensional data. *Statistics in Medicine*, 40(27):6178–6196, 2021. [p2, 3, 4, 5, 16, 17]

N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for Cox's proportional hazards model via coordinate descent. *Journal of statistical software*, 39(5):1, 2011. [p5]

E. W. Steyerberg. *Clinical Prediction Models*. Springer, 2009. [p1]

T. M. Therneau and P. M. Grambsch. Modeling survival data: extending the Cox model. 2000. [p1]

H. C. Van Houwelingen. Dynamic prediction by landmarking in event history analysis. *Scandinavian Journal of Statistics*, 34(1):70–85, 2007. [p1]

P. J. Verweij and H. C. Van Houwelingen. Penalized likelihood in Cox regression. *Statistics in Medicine*, 13(23-24):2427–2436, 1994. [p4]

H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B*, 67(2):301–320, 2005. [p11]

Mirko Signorelli
Mathematical Institute, Leiden University
Niels Bohrweg 1, 2333 CA Leiden (NL)
https://mirkosignorelli.github.io
ORCiD: 0000-0002-8102-3356
msignorelli.papers@gmail.com