Individual claims reserving with ReSurv

Munir Hiabu[1], Emil Hofman[2], and Gabriele Pittarello[3]

[1]Department for Mathematical Sciences, University of Copenhagen

[2]Department for Mathematical Sciences, University of Copenhagen

[3]ESOMAS department, University of Turin

August 5, 2024

Abstract

For non-life reserving, the industry typically relies on chain-ladder-type methods based on development triangles stemming from an aggregation of individual claims data. The more detailed databases that make up the development triangles are often held by insurers and contain information about claims at an individual level that could potentially improve reserving. In this manuscript we present `ReSurv`, an R package for modelling feature-dependent development factors using individual reserving data. These development factors can be used to predict claim frequencies. The methodology implemented in the package was derived in Hiabu et al. (2023). This paper includes an informal presentation of our framework and a detailed, practitioner-oriented guide to the use of our package.

Keywords: survival analysis; individual reserving; non-life insurance; proportional hazard.

1 Introduction

In the non-life insurance practice, the term *reserves* is used to indicate the provisions of a non-life insurer (Friedland, 2010, p. 13). Among these provisions, we find the reserve for Incurred But Not Reported (IBNR) claims. Since the number of IBNR claims is not known at the time the reserve is computed, actuaries must estimate it. One option is to model the frequency and severity of IBNR claims separately. In this paper we present a method for estimating the number of claims. Estimating the size of IBNR claims is outside the scope of our project.

Modelling of future reports is usually utilizing development triangles, an aggregate representation of the reserving data that is standard practice in the industry (Brown et al., 2023). A development triangle is a set of observations

$$\mathcal{O} = \{O_{kj} : k, j = 0, \dots, m; k + j \leqslant m\}, \quad m > 0,$$

where k indicates the accident date and j indicates the development date up to a maximum date m. Here, O_{kj} as the total number of reported claims in accident date k and development date j. A commonly used model with development triangles is the chain-ladder, which uses the so-called development factors,

$$\hat{f}_{kj} = \sum_{l \leqslant j} O_{kl} / \sum_{l < j} O_{kl}$$

for predictions. While reserving models for aggregate data have a long history in the industry, the advent of new advanced computing tools has encouraged actuarial professionals to explore reserving models based on individual data (Richman, 2021). Often the more advanced techniques are based on sound theory, but there is no open source implementation that makes the models directly applicable in practice. A notable exception is the hierarchical model in Crevecoeur, Antonio, et al. (2023), implemented in the R package `hirem` (Crevecoeur and Robben, 2024). In this manuscript we present a pipeline for individual claims reserving using the R package `ReSurv`. Our package implements a feature-dependent estimator of the claim development that can be used to predict future Incurred But Not Reported (IBNR) claims. In the next section, we provide a simplified explanation of the methodology in Hiabu et al. (2023); for a full review, please refer to the main manuscript. After describing in detail how to install the package, we discuss a pipeline for individual reserving in Section 2. We show a data application on a simulated dataset in Section 3.

1.1 Hiabu et al., 2023 in a nutshell

The mathematical basis for `ReSurv` follows from the formulation of the reserving problem in a survival analysis setting. We model the reporting delay of individual claims by specifying a regression model for the corresponding hazard function. Consider a set of reported claims. Each observed claim is associated with a reporting date t_i, an accident date u_i and a set of covariates $x_i \in \mathbb{R}^p$. We specify the following regression model for the hazard rate

$$\alpha(t|u,x) = \alpha_0(t)e^{\phi(x,u;\theta)}, \tag{1}$$

where $\alpha_0(t)$ is called the baseline hazard and $e^{\phi(x,u;\theta)}$ is the risk score; a component that depends on the features x_i and the accident period $u_{(k)}$ and some parameters θ. Estimation of the risk score $\hat{\phi}\,(x,u;\theta)$ can be performed in `ReSurv` with three different algorithms: the cox model (COX, Cox, 1972), neural networks (NN, Katzman et al., 2018) and gradient boosting (XGB, Chen and Guestrin, 2016).

- In Cox (1972), the risk score function is assumed to be linear, $\phi\,(x,u;\theta) = \theta^T x + \theta_u u$, with $\theta \in \mathbb{R}^p$ and $\theta_u \in \mathbb{R}$. Our package provides the option to include splines for modelling continuous features.
- In NN, the parameter θ represents the weights of a feed-forward neural network.
- In XGB, the log-risk function is an ensemble of decision trees, i.e., functions piecewise constant on rectangles.

After an estimator for $\hat{\phi}\,(x,u;\theta)$ is derived, the baselines $\alpha_0(t)$ is estimated using the full-likelihood where it is assumed that claim reports are uniformly distributed within a tie. Putting the estimators together we derive an estimator of the hazard function

$$\hat{\alpha}(t|u,x) = \hat{\alpha}_0(t)e^{\hat{\phi}(x,u;\theta)},$$

Finally, we model the development factor from development period $j-1$ to j and accident period k as

$$\tilde{f}_{kj}(x) = \frac{2 + \hat{\alpha}(t^{(j)}|u^{(k)},x)}{2 - \hat{\alpha}(t^{(j)}|u^{(k)},x)}, \tag{2}$$

where $t^{(j)}, u^{(k)}$ are the development time and accident time corresponding to the jth development date and kth accident date respectively. From here, these development factors can be applied using the chain-ladder rule for forecasting. By introducing feature and accident period dependency in the hazard estimation, we allow the same properties in the development factors.

Installation

The developer version of `ReSurv` can be installed from GitHub.

```
> library(devtools)
> devtools::install_github('edhofman/resurv')
```

The package can then be imported in R using the command

```
> library(ReSurv)
```

Additional resources on our project can be found at https://github.com/edhofman/ReSurv. We remark that this manuscript refers to version `0.0.2` of our package:

```
> packageVersion("ReSurv")
0.2
```

2 IBNR modelling using ReSurv

In this section, we illustrate individual claims reserving in six steps that simulate the steps that an actuary can take to perform individual reserving using our software.

1. **Start from the data**. In this vignette we will use the simulator embedded in our package to generate synthetic reserving data.
2. **Pre-process the data**. Before using a reserving model, individual data must be elaborated in a format that is ready for using the individual model. This step involves the one-hot encoding of categorical features and scaling of continuous features.
3. **Choose a model**. As mentioned earlier, our package allows us to model the log-risk function using three different models (COX, NN and XGB). The aim of Section 3 is to compare the three available approaches performances according to some performance metrics that we will define.
4. **Optimise the model hyperparamters**. Machine learning algorithms are sensitive to the hyperparameters chosen. In our package we have a basic function to perform K-fold cross validation, which can be combined with the package `BayesianParOptimisation` for a more sophisticated routine.

5. **Estimate the model parameters.** Once we have estimated the hyperparameters of the model, we can optimise the parameters θ.

6. **Visualize claim development and predict future IBNR.** The optimised model can be used to plot accident date and feature dependent development factors as well as predict IBNR claims for different data granularities, as explained in Hiabu et al. (2023). For example, if actuaries have daily data available for fitting, our approach is flexible enough to report and visualize future claims on a monthly, quarterly or annual scale.

2.1 Data simulation

Let us consider the `data_generator` function.

```
> input_data <- data_generator(random_seed = 7,
                               scenario='alpha',
                               time_unit = 1/360,
                               years = 4,
                               period_exposure = 200)
```

This function contains 5 different parameters:

- The `random_seed`, to guarantee full replicable code.
- Our package offers 5 different simulated scenarios that can be used to replicate our manuscript analysis. The scenarios are described below and selected with the parameter `scenario`. Here, we choose to simulate from the so-called scenario Alpha.
- `time_unit` controls the data granularity (time unit) as a fraction of a year. In our manuscript we generate daily data and set `time_unit`$= 1/360$.
- `years` is the total number of accident years in the simulation. In the manuscript we use a four years time horizon for daily data.
- The `period_exposure` consists of the total claims exposure for time unit. In our manuscript we have a daily exposure of 200 claims.

Our package allows to simulate reserving data under 5 scenarios using the function `data_generator`. The simulator is based on the `SynthETIC` package (Avanzi et al., 2021). We named the 5 scenarios

Feature	Description
`claim_number`	Policy identifier.
`claim_type`	Type of claim.
`AP`	Accident period.
`RP`	Reporting period.
`DP`	Development period.
`DP_rev`	Reverse time development period (`years`-`DP`)
`TR`	Truncation time
`I`	Indicator, if reported equal to one
`AT`	Accident date (in continuous time).
`RT`	Reporting period (in continuous time).
`DT`	Development period (in continuous time).

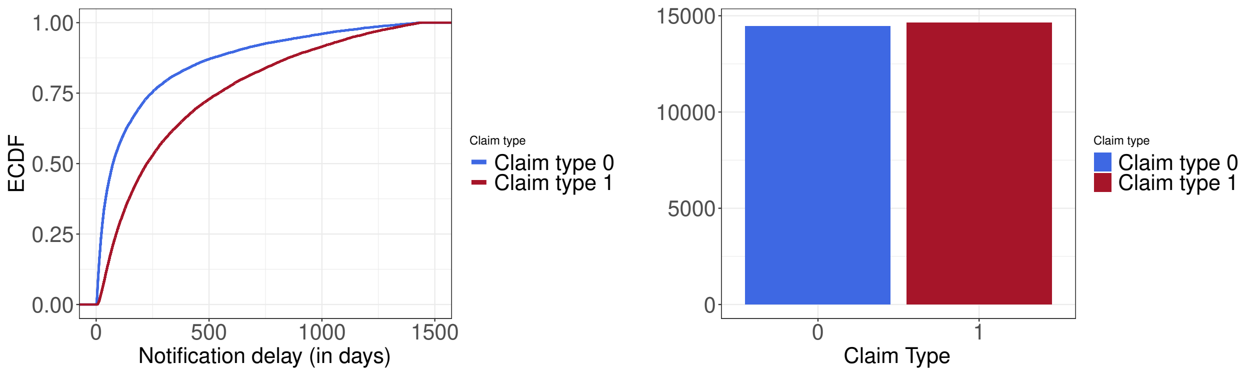
Table 1: Description of the information in the simulated data.

Alpha, Beta, Gamma, Delta, Epsilon. The 5 scenarios contain the information described in Table 1. Our simulated data are constituted of a mix of short tail claims (`claim_type 0`) and claims with longer resolution (`claim_type 1`). We chose the parameter of the simulator to resemble a mix of property damage (`claim_type 0`) and bodily injuries (`claim_type 1`).

However, each scenario has distinctive characteristics. Scenario Alpha is a mix of `claim_type 0` and `claim_type 1` with same number of claims volume at each accident period. Differently from scenario Alpha, in scenario Beta the volumes of `claim_type 1` are decreasing in the most recent accident periods. In scenario Gamma we add an interaction between `claim_type 1` and accident period: in a real world setting this can be motivated by a change in consumer behavior or company policies resulted in different reporting patterns over time. In scenario Delta, we introduce a seasonality effect dependent on the accident period for `claim_type 0` and `claim_type 1`. In the real word, scenario Delta resembles seasonal changes in the workforce composition.

The object `input_data` is a `tibble data.frame` and its structure can be displayed with the following command.

```
> str(input_data)
 tibble [29,088 × 12] (S3: tbl_df/tbl/data.frame)
  $ claim_number: int [1:29088] 1 2 3 4 5 6 7 8 9 10 ...
  $ AT          : num [1:29088] 0.5227 0.6338 0.4861 0.3622 0.0711 ...
  $ RT          : num [1:29088] 18.8 200.3 894.9 882.5 35.1 ...
  $ claim_type  : num [1:29088] 0 0 0 0 0 0 0 0 0 0 ...
  $ AP          : num [1:29088] 1 1 1 1 1 1 1 1 1 1 ...
```



(a) Empirical Cumulative Density Function, notification delay (`RT`). (b) Data distribution by claim type (`claim_type`).

Figure 1: Empirical simulated cumulative distribution function (left-hand side) and data distribution by claim type (right-hand side).

```
 $ RP            : num [1:29088] 19 201 895 883 36 ...
 $ DT            : num [1:29088] 18.2 199.7 894.4 882.1 35 ...
 $ DP            : num [1:29088] 19 201 895 883 36 ...
 $ DP_rev        : num [1:29088] 1422 1240 546 558 1405 ...
 $ DT_rev        : num [1:29088] 1422 1240 546 558 1405 ...
 $ TR            : num [1:29088] 0 0 0 0 0 0 0 0 0 0 ...
 $ I             : num [1:29088] 1 1 1 1 1 1 1 1 1 1 ...
```

We show the empirical cumulative density function of the simulated notification delay (Figure 1a) and the data distribution by claim type (Figure 1b). We can notice quicker notification for `claim_type 0`. The two claim types have similar volumes.

2.2 Data pre-processing

Before fitting our model on the data, the individual data must be pre-processed. In our software, this can be achieved with the built-in function `IndividualDataPP`. `IndividualDataPP` creates an individual data set that is ready to be modelled.

```
> individual_data <- IndividualDataPP(data = input_data,
                                      categorical_features = c("claim_type"),
                                      continuous_features = "AP",
                                      accident_period="AP",
                                      calendar_period="RP",
```

```
                    input_time_granularity = "days",
                    output_time_granularity = "quarters",
                    years=4)
```

- The input `data`. Here, the simulated output of `data_generator` in Section 2.1.
- `categorical_features` and `continuous_features` allows us to specify which columns to handle as categorical, and which as numeric. This can be important in a NN setting when doing preprocessing of data. We only have one categorical feature in our simulated data, but multiple can be specified.
- `accident_period` and `calender_period` tells us the name of the corresponding columns in the `data`.
- The `input_time_granularity` tells us how granular our input is, and `output_time_granularity` allows the user to specify the level of the output development factors.
- `years` is the total number of accident years in the `data`.

The output of `IndividualDataPP` is a list containing:

- `full.data`: the input data after pre-processing.
- `starting.data`: the input data as they were provided from the user.
- `string_formula_i`: string of the `survival` formula to model the data in input granularity.
- `training.data`: the input data pre-processed for training.
- `conversion_factor`: the conversion factor for going from input granularity to output granularity. E.g, the conversion factor for going from months to quarters is 1/3.
- `string_formula_o`: string of the `survival` formula to model the in data output granularity.
- `continuous_features`: the continuous features names as provided from the user.
- `categorical_features`: the categorical features names as provided from the user.

After pre-processing, we provide a standard encoding for the time components. This regards the output in `training.data` and `full.data`. In the `ReSurv` notation:

- `AP_i`: input accident period.
- `AP_o`: output accident period.
- `DP_i`: input development period in forward time.
- `DP_rev_i`: input development period in reverse time.
- `DP_rev_o`: output development period in reverse time.
- `TR_i`: input truncation time.
- `TR_o`: output truncation time.
- `I`: event indicator, under this framework is equal to one for each entry.

2.3 Selection of the hyper-parameters

`ReSurv` offers a built-in implementation of a standard K-Fold cross-validation (Hastie et al., 2009): the `ReSurvCV` method of an `IndividualDataPP` object. We show an illustrative example for XGB and NN below.

In Section 2.4, we will show how to combine `ReSurvCV` to the methods from Snoek et al., 2012 implemented in the R package `ParBayesianOptimization` (Wilson, 2022).

2.3.1 XGB: K-Fold cross-validation

`ReSurvCV` requires three inputs:

- The input `IndividualDataPP`, will be the output from the data pre-processing step.
- Three different `models` are allowed to estimate the hazard function. Here we chose XGB, the other possibilities include NN (deep survival neural network) and COX.
- To perform the grid search, we need to specify the `hyperparameter_grid` on which we optimize. This will be dependent on the chosen model, and in this example we have given values for some parameters for XGB.

We remark that our XGB implementation extends for left-truncation and tied data the implementation from Chen and Guestrin, 2016. For a more detailed description of the model parameters that can be cross-validated for XGB, please visit `xgboost` reference guide.

```
> resurv.cv.xgboost <- ReSurvCV(IndividualDataPP=individual_data,
                                model="XGB",
                                hparameters_grid=list(booster="gbtree",
                                                      eta=c(.001,.01,.2,.3),
                                                      max_depth=c(3,6,8),
                                                      subsample=c(1),
                                                      alpha=c(0,.2,1),
                                                      lambda=c(0,.2,1),
                                                      min_child_weight=c(.5,1)),
                                print_every_n = 1L,
                                nrounds=500,
                                verbose=F,
                                verbose.cv=T,
                                early_stopping_rounds = 100,
                                folds=5,
                                parallel=T,
                                ncores=2,
                                random_seed=1)
```

For XGB, the output of `ReSurvCV` consists in two `data.frame`: `out.cv` and `out.cv.best.oos`. The two outputs contain the hyperparameters `booster`, `eta`, `max_depth`, `subsample`, `alpha`, `lambda`, `min_child_weight`. They also contain the metrics `train.lkh` (in-sample likelihood), `test.lkh` (out-of-sample likelihood), and the computational time `time`. `out.cv` contains the output of the cross-validation (all the input parameters combinations). `out.cv.best.oos` contains the combination with the best out of sample likelihood.

2.3.2 NN: K-Fold cross-validation

The `ReSurv` NN implementation uses `reticulate` to interface R Studio to Python and it is based on a similar approach to Katzman et al. (2018), corrected to account for left-truncation and ties in the data. Similarly to the original implementation we relied on the Python library `pytorch` (Paszke et al., 2019). The syntax of our NN is then the syntax of `pytorch`. See the reference guide for further information on the NN parametrization.

```
> resurv.cv.nn <- ReSurvCV(IndividualDataPP=individual_data,
                      model="NN",
                      hparameters_grid=list(num_layers = c(1,2),
                                            num_nodes = c(2,4),
                                            optim="Adam",
                                            activation = "ReLU",
                                            lr=.5,
                                            xi=.5,
                                            eps = .5,
                                            tie = "Efron",
                                            batch_size = as.integer(5000),
                                            early_stopping = 'TRUE',
                                            patience  = 20
                      ),
                      epochs=as.integer(300),
                      num_workers = 0,
                      verbose=F,
                      verbose.cv=T,
                      folds=3,
                      parallel=F,
                      random_seed = as.integer(Sys.time()))
```

For NN models, the columns in `out.cv` and `out.cv.best.oos` are the hyperparameters `num_layers`, `optim`, `activation`, `lr`, `xi`, `eps`, `tie`, `batch_size`, `early_stopping`, `patience`, `node` train.lkh test.lkh. They also contain the metrics `train.lkh`, `test.lkh`, and the computational time `time`.

2.4 `ReSurv` and Bayesian Parameters Optimisation

Our methods can be easily combined with those from the `ParBayesianOptimization` package. While we refer to Snoek et al., 2012 for a mathematical explanation of the Bayesian Optimisation method that we use. We show a code example below.

2.4.1 XGB: Bayesian Parameters Optimisation

We specify the bounds of our parameters search.

```
> bounds <- list(eta = c(0, 1),
             max_depth = c(1L, 25L),
             min_child_weight = c(0, 50),
             subsample = c(0.51, 1),
             lambda = c(0, 15),
             alpha = c(0, 15))
```

Secondly, we need to specify an objective function to be optimized with the Bayesian approach. The score metric we inspect is the negative (partial) likelihood. The likelihood is returned with negative sign as Wilson (2022) is maximizing the objective function.

```
> obj_func <- function(eta,
                  max_depth,
                  min_child_weight,
                  subsample,
                  lambda,
                  alpha) {


xgbcv <- ReSurvCV(IndividualDataPP=individual_data,
                  model="XGB",
                  hparameters_grid=list(booster="gbtree",
                                        eta=eta,
                                        max_depth=max_depth,
                                        subsample=subsample,
                                        alpha=lambda,
                                        lambda=alpha,
                                        min_child_weight=min_child_weight),
                  print_every_n = 1L,
                  nrounds=500,
                  verbose=F,
```

```
                    verbose.cv=T,
                    early_stopping_rounds = 30,
                    folds=3,
                    parallel=F,
                    random_seed = as.integer(Sys.time()))

lst <- list(

  Score = -xgbcv$out.cv.best.oos$test.lkh,
  train.lkh = xgbcv$out.cv.best.oos$train.lkh
)

return(lst)
```

As a last step, we use the `bayesOpt` function to perform the optimization.

```
> bayes_out <- bayesOpt(
  FUN = obj_func
  , bounds = bounds
  , initPoints = 50
  , iters.n = 1000
  , iters.k = 50
  , otherHalting = list(timeLimit = 18000)
)
```

To select the optimal hyperparameters we inspect `bayes_out$scoreSummary` output in Table 2. Below we print the first five rows of one of our runs. Observe `scoreSummary` is a `data.table` that also contains some parameters specific of the original implementation (see Wilson (2022) for more details)

We select the final combination that minimizes the negative (partial) likelihood, in the 'Score' column.

Epoch	Iteration	...	num_layers	num_nodes	optim	activation	lr	xi	eps	batch_size	Elapsed	Score	train.lkh
0	1	...	9	8	1	2	0.08	0.35	0.03	1226	6094.91	-6.24	6.28
0	1	...	9	2	2	1	0.47	0.50	0.10	3915	7307.31	-7.27	7.30
0	1	...	9	9	2	1	0.40	0.49	0.18	196	6719.70	-5.98	5.97
0	1	...	8	8	1	2	0.03	0.23	0.01	4508	8893.46	-7.39	7.41
0	1	...	9	7	2	1	0.13	0.13	0.12	900	3189.15	-6.21	6.23

Table 2: Header of the output of `bayes_out$scoreSummary`. We select the hyperparameters that minimise the negative log-likelihood in the column `Score`.

NN: Bayesian Parameters Optimisation

Similarly to the XGB case, we specify the bounds for the hyper-parameters search in the NN case.

```
> bounds <- list(num_layers = c(2L,10L),
             num_nodes = c(2L,10L),
             optim=c(1L,2L),
             activation = c(1L,2L),
             lr=c(.005,0.5),
             xi=c(0,0.5),
             eps = c(0,0.5)
             )
```

We then define an objective function.

```
>  obj_func <- function(num_layers,
                   num_nodes,
                   optim,
                   activation,
                   lr,
                   xi,
                   eps) {


optim = switch(optim,
              "Adam",
              "SGD")
activation = switch(activation, "LeakyReLU","SELU")
batch_size=as.integer(5000)
```

```
number_layers=as.integer(num_layers)
num_nodes=as.integer(num_nodes)

deepsurv_cv <- ReSurvCV(IndividualData=individual_data,
                  model="NN",
                  hparameters_grid=list(num_layers = num_layers,
                                        num_nodes = num_nodes,
                                        optim=optim,
                                        activation = activation,
                                        lr=lr,
                                        xi=xi,
                                        eps = eps,
                                        tie = "Efron",
                                        batch_size = batch_size,
                                        early_stopping = 'TRUE',
                                        patience  = 20
                  ),
                  epochs=as.integer(300),
                  num_workers = 0,
                  verbose=F,
                  verbose.cv=T,
                  folds=3,
                  parallel=F,
                  random_seed = as.integer(Sys.time()))



lst <- list(

  Score = -deepsurv_cv$out.cv.best.oos$test.lkh,

  train.lkh = deepsurv_cv$out.cv.best.oos$train.lkh
)
```

```
return(lst)
```

The optimisation is then performed with the `bayesOpt` function as follows.

```
> bayes_out <- bayesOpt(FUN = obj_func,
  bounds = bounds,
  initPoints = 50,
  iters.n = 1000,
  iters.k = 50,
  otherHalting = list(timeLimit = 18000))
```

2.5 Estimation

Once we have found on our data the optimal hyper-parameters for NN and XGB we can our algorithms to estimate the parameters θ.

COX

```
> resurv.fit.cox <- ReSurv(individual_data,
                   hazard_model = "COX")
```

The `ReSurv` fit output is a list containing

- `model.out`: `list` containing the pre-processed covariates data for the fit (`data`) and the basic model output (`model.out`;COX, XGB or NN).
- `is_lkh`: `numeric` Training negative log likelihood.
- `os_lkh`: `numeric` Validation negative log likelihood. Not available for COX.
- `hazard_frame`: `data.frame` containing the fitted log-risk and baseline (`expg`, `baseline`), the fitted hazard (`hazard`), the fitted development factors (`dev_f_i`), their cumulative version (`cum_dev_f_i`), the fitted survival function `S_i`

- `hazard_model`: `string` chosen hazard model (COX, NN or XGB).
- `IndividualDataPP`: starting `IndividualDataPP` object.

XGB

After selecting the hyper parameters we can finally fit our models to our pre-processed data. The optimised hyper-parameters are saved in `hparameters_xgb` as a list.

```
> hparameters_xgb = list(params=list(booster="gbtree",
                         eta=0.9611239,
                         subsample=0.62851,
                         alpha= 5.836211,
                         lambda=15,
                         min_child_weight=29.18158,
                         max_depth = 1),
             print_every_n = 0,
             nrounds=3000,
             verbose=F,
             early_stopping_rounds = 500)
```

In the `ReSurv` package the fitting can be performed using the homonymous `ReSurv` method.

```
> resurv.fit.xgb <- ReSurv(individual_data,
                  hazard_model = "XGB",
                  hparameters = hparameters_xgb)
```

The `ReSurv` function, simply requires to specify the pre-processed individual data, the selected model for the hazard (`hazard_model` argument) and the necessary hyperparameters (`hparameters` argument).

NN

In the NN case we find the following hyper-parameters.

```
> hparameters_nn = list(num_layers= 2,
          early_stopping= TRUE,
          patience=350,
          verbose=FALSE,
          network_structure=NULL,
          num_nodes= 10,
          activation ="LeakyReLU",
          optim ="SGD",
          lr =0.02226655,
          xi=0.4678993,
          epsilon= 0,
          batch_size= 5000,
          epochs= 5500,
          num_workers= 0,
          tie="Efron" )
```

We can fit our NN model as follows.

```
> resurv.fit.nn <- ReSurv(individual_data,
                   hazard_model = "NN",
                   hparameters = hparameters_nn)
```

2.6 Prediction

We use the method `predict` to predict the future claim frequencies. The method can be used by simply specifying a `ReSurv` model. Below, we only show an example for the COX model but a similar routine can be used for NN and XGB.

```
> resurv.fit.predict.Q <- predict(resurv.fit.cox)
```

Our software also allows to predict IBNR for different granularities. In order to do so, it is sufficient to pre-process the input data using the `IndividualDataPP` class and changing the `output_time_granularity` argument. In the example below, we process our data for yearly predictions.

```
> individual_dataY <- IndividualDataPP(input_data,
                                id="claim_number",
                                continuous_features="AP_i",
                                categorical_features="claim_type",
                                accident_period="AP",
                                calendar_period="RP",
                                input_time_granularity = "days",
                                output_time_granularity = "years",
                                years=4,
                                continuous_features_spline=NULL,
                                calendar_period_extrapolation=F)
```

We then use the `predict` method on the new data for forecasting.

```
> resurv.fit.predict.Y <- predict(resurv.fit.cox,
                               newdata=individual_dataY,
                               grouping_method = "probability")
```

The same routine is applied monthly in the next code.

```
> individual_dataM <- IndividualDataPP(input_data,
                                  id="claim_number",
                                  continuous_features="AP_i",
                                  categorical_features="claim_type",
                                  accident_period="AP",
                                  calendar_period="RP",
                                  input_time_granularity = "days",
                                  output_time_granularity = "months",
                                  years=4,
                                  continuous_features_spline=NULL,
                                  calendar_period_extrapolation=F)
```

```
> resurv.fit.predict.M <- predict(resurv.fit.cox,
```

```
                                          newdata=individual_dataM,
                                          grouping_method = "probability")
```

A summary of the predictions total output can be displayed with a print of the predictions `summary`.

```
> model_s <- summary(resurv.fit.predict.Y)
> print(model_s)
  Hazard model:
 "COX"

 Categorical Features:
 claim_type
 Continuous Features:
 AP_i
 Total IBNR level:
 [1]  5480
```

Using our approach we can produce, for each combination of features, the feature-dependent development factors in Equation (2). In Figure 2 we produce an example for the COX output illustrated in this Section and compare it with the chain-ladder model. We show for monthly, quarterly and yearly data the development factors fitted with the COX model for some combinations of features (rows two and three). Differently from the chain-ladder development factors (row one), we can catch data heterogeneity by feature.

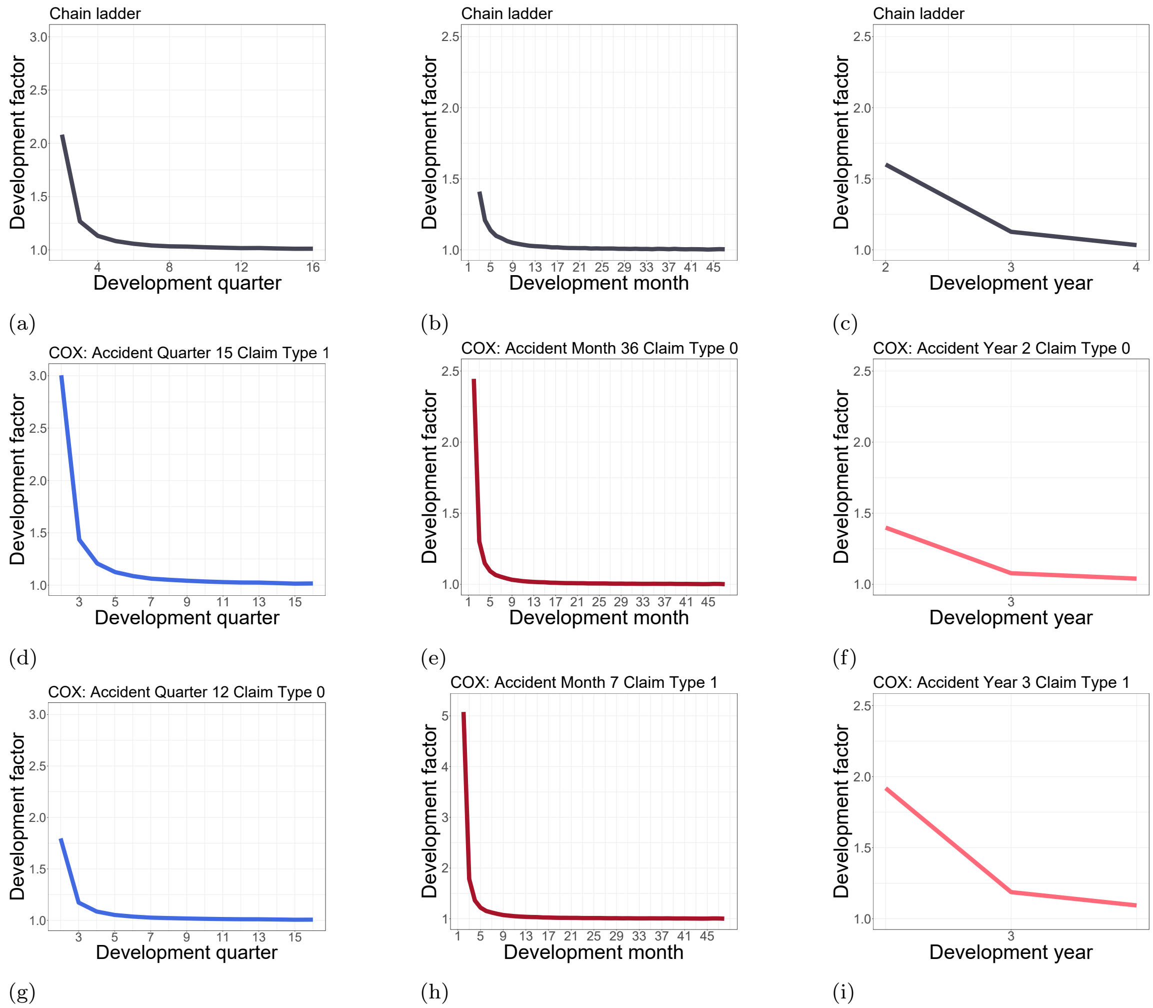


Figure 2: The first column, shows quarterly development factors for the chain ladder (top panel) and the COX model for feature combinations Accident Quarter 15 and `Claim_type 0` (center panel) and Accident Quarter 12 and `Claim_type 1` (bottom panel). The second column shows monthly development factors for the chain ladder (top panel) and the COX model for the feature combinations Accident Month 36 and `Claim_type 1` (top panel) and Accident Month 7 and `Claim_type 1` (bottom panel). The third column shows yearly development factors for the chain ladder (top panel) and the COX model for the feature combinations Accident Year 2 and `Claim_type 0` (central panel), and Accident Quarter 3 and `Claim_type 1` (bottom panel).

3 Data application

In this Section, we show a small data application that compares our fitted models on the data from Section 2.1 using three different performance metrics defined below. The code is shown for the COX model but similar compuations can be performed with XGB and NN. The three metrics are the Total Absolute Relative Error ($\texttt{ARE}^{\texttt{TOT}}$), Calendar Absolute Relative Error ($\texttt{ARE}^{\texttt{CAL}}$), and the Continuously Ranked Probability Score (CRPS; Gneiting, A. Raftery, et al., 2004). While the code for the first two metrics is computed in Appendix C, we show here the usage of the built-in function survival_crps for the computation of the CRPS of ReSurv models.

Total Absolute Relative Error $\texttt{ARE}^{\texttt{TOT}}$

We first evaluate the Total (relative) Absolute Errors $\texttt{ARE}^{\texttt{TOT}}$ on the input grid. The $\texttt{ARE}^{\texttt{TOT}}$ is defined as

$$\texttt{ARE}^{\texttt{TOT}} = \frac{\sum_{j,k:k+j>m} |\sum_x O_{k,j}(x) - \sum_x \hat{O}_{k,j}(x)|}{\sum_{j,k:k+j>m} \sum_x O_{k,j}(x)}, \tag{3}$$

where $\hat{O}_{k,j}(x)$ denotes the estimated reportings in accident period k and development period j. The $\texttt{ARE}^{\texttt{TOT}}$ computation for the COX fit is displayed in Appendix C

Calendar Absolute Relative Error $\texttt{ARE}^{\texttt{CAL}}$

We then want to evaluate our models performance a second time diagonal-wise with a different performance metric. We considers the new information available at the end of each calendar period until development. We call this metric the Total (relative) Absolute Errors by Calendar time ($\texttt{ARE}^{\texttt{CAL}}$). The $\texttt{ARE}^{\texttt{CAL}}$ is defined as

$$\texttt{ARE}^{\texttt{CAL}} = \frac{\sum_{\tau=m+1}^{2m-1} \sum_{j,k:k+j=\tau} |\sum_x O_{k,j}(x) - \sum_x \tilde{f}_{k,j}(x) O_{k,j-1}(x)|}{\sum_{j,k:k+j>m} \sum_x O_{k,j}(x)}. \tag{4}$$

The computation of the quarterly $\texttt{ARE}^{\texttt{CAL}}$ for the COX model is shown in Appendix C.

Continuously Ranked Probability Score (CRPS)

The Continuously Ranked Probability Score (CRPS) is defined in Gneiting and A. E. Raftery, 2007 as

$$\begin{aligned}
\mathrm{CRPS}(\hat{S}(z|X,U), y) &= \int_0^\infty \mathrm{PS}(\hat{S}(z|X,U), \mathbb{I}\{y > z\}) \mathrm{d}z \\
&= \int_0^\infty (\hat{S}(z|X,U) - \mathbb{I}\{y > z\})^2 \mathrm{d}z \\
&= \int_0^y \left(1 - \hat{S}(z|X,U)\right)^2 dz + \int_y^{+\infty} \left(\hat{S}(z|X,U)\right)^2 dz,
\end{aligned}$$

Performance Metric	Chain-Ladder	**COX**	**NN**	**XGB**
$\texttt{ARE}^{\texttt{TOT}}$	0.115	0.116	**0.113**	0.124
$\texttt{ARE}^{\texttt{CAL}}$	0.111	0.107	**0.105**	0.111
CRPS (average)	-	366.264	**365.229**	367.950

Table 3: Results in terms of $\texttt{ARE}^{\texttt{TOT}}$, $\texttt{ARE}^{\texttt{CAL}}$ and CRPS (rows) for the different models (columns) on the simulated data set.

with $\mathrm{PS}(\hat{S}(z|X,U_i), \mathbb{I}\{y > z\}) = (\hat{S}(z|X,U) - \mathbb{I}\{y > z\})^2$ being the Brier Score Selten, 1998; Gneiting and A. E. Raftery, 2007.

We can use correspondence between survival function and predicted development factors (Hiabu et al., 2023)

$$\hat{S}(j|X,U) = \frac{1}{\prod_{l=1}^{j} \hat{f}_{k,l}(x)}. \tag{5}$$

The CRPS can be computed with the built-in method `survival_crps`

```
> crps <- survival_crps(resurv.fit.cox)
```

The output of `survival_crps` is a `data.table` that contains the CRPS (`crps`) for each observation (`id`). For comparison among models, we take the average CRPS on the data.

```
> m_crps <- mean(crps$crps)
> m_crps
366.2639
```

Models comparison

The results on the data simulated in Section 2.1 are shown in Table 3. In our numerical application, we seem to prefer the NN model to COX and XGB according to the metrics defined in this section. The results are shown in bold for the NN model. We observe that the NN model shows the smallest $\texttt{ARE}^{\texttt{TOT}}$ and $\texttt{ARE}^{\texttt{CAL}}$, and smallest average CRPS. However, we observe that NN and XGB can be sensitive to hyper-parameters choice and they require the extensive cross-validation procedure that we illustrated in the previous sections. Our models are compared with the Chain-Ladder (column one). While the Chain-Ladder seems to fine, it is outperformed by the NN model.

Further Reading

The interested reader can find additional resources on the package at

https://github.com/edhofman/ReSurv.

The ArXiv version of the main manuscript has archive identifier

arxiv:2312.14549.

An `RMarkdown` version of the code included in this manuscript can be found at

https://github.com/edhofman/ReSurv/blob/main/vignettes/cas_call.Rmd.

References

Avanzi, B., Taylor, G., Wang, M., and Wong, B. (2021). "SynthETIC: An individual insurance claim simulator with feature control". In: *Insurance: Mathematics and Economics* **100**, pp. 296–308.

Brown, B. Z., Julga, L., and Merz, J. (2023). "Best Practices for Property and Casualty Actuarial Reserving Departments". In: *CAS E-Forum*.

Chen, T. and Guestrin, C. (2016). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: `10.1145/2939672.2939785`. URL: `http://doi.acm.org/10.1145/2939672.2939785`.

Cox, D. R. (1972). "Regression models and life-tables". In: *Journal of the Royal Statistical Society: Series B (Methodological)* **34**.2, pp. 187–202.

Crevecoeur, J., Antonio, K., Desmedt, S., and Masquelein, A. (2023). "Bridging the gap between pricing and reserving with an occurrence and development model for non-life insurance claims". In: *ASTIN Bulletin: The Journal of the IAA* **53**.2, pp. 185–212.

Crevecoeur, J. and Robben, J. (2024). *hirem: Hierarchical reserving models*. R package version 0.1.0.

Friedland, J. (2010). "Estimating unpaid claims using basic techniques". In: *Casualty Actuarial Society*. Vol. 201. 0.

Gneiting, T. and Raftery, A. E. (2007). "Strictly proper scoring rules, prediction, and estimation". In: *Journal of the American statistical Association* **102**.477, pp. 359–378.

Gneiting, T., Raftery, A., Balabdaoui, F., and Westveld, A. (2004). "Verifying probabilistic forecasts: Calibration and sharpness". In: *Preprints, 17th conf. on probability and statistics in the atmospheric sciences, seattle, wa, amer. meteor. soc.* Vol. 2.

Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction.* Vol. 2. Springer.

Hiabu, M., Hofman, E., and Pittarello, G. (2023). "A machine learning approach based on survival analysis for IBNR frequencies in non-life reserving". In: *arXiv preprint arXiv:2312.14549*.

Katzman, J. L., Shaham, U., Cloninger, A., Bates, J., Jiang, T., and Kluger, Y. (2018). "DeepSurv: personalized treatment recommender system using a Cox proportional hazards deep neural network". In: *BMC medical research methodology* **18**.1, pp. 1–12.

Paszke, A. et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Richman, R. (2021). "AI in actuarial science–a review of recent advances–part 2". In: *Annals of Actuarial Science* **15**.2, pp. 230–258.

Selten, R. (1998). "Axiomatic characterization of the quadratic scoring rule". In: *Experimental Economics* **1**, pp. 43–61.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems* **25**.

Wickham, H. et al. (2019). "Welcome to the tidyverse". In: *Journal of Open Source Software* **4**.43, p. 1686. DOI: 10.21105/joss.01686.

Wilson, S. (2022). *ParBayesianOptimization: Parallel Bayesian Optimization of Hyperparameters*. R package version 1.2.6. URL: https://CRAN.R-project.org/package=ParBayesianOptimization.

A Code to replicate Figure 1

The code in this section relies on the libraries `dplyr` and `ggplot2`

```
> library(dplyr)
> library(ggplot2)
```

A.1 Code to replicate Figure 1a

```
> p1 <- input_data %>%
          as.data.frame() %>%
          mutate(claim_type=as.factor(claim_type))%>%
          ggplot(aes(x=RT-AT, color=claim_type)) +
          stat_ecdf(size=1) +
          labs(title="",
          x="Notification delay (in days)",
          y="ECDF") +
          xlim(0.01,1500)+
          scale_color_manual(values=c("royalblue", "#a71429"),
          labels=c("Claim type 0","Claim type 1")) +
          scale_linetype_manual(values=c(1,3),
          labels=c("Claim type 0","Claim type 1"))+
          guides(color = guide_legend(title="Claim type",
          override.aes = list(color = c("royalblue", "#a71429"),
          linewidth = 2)),
          linetype = guide_legend(title="Claim type",
          override.aes = list(linetype = c(1,3),
          linewidth = 0.7))) +
          theme_bw()+
          theme(axis.text=element_text(size=20),
          axis.title.y = element_text(size=20),
          axis.title.x  = element_text(size=20),
          legend.text = element_text(size=20))
> p1
```

A.2 Code to replicate Figure 1b

```
> p2<- input_data %>%
          as.data.frame() %>%
          mutate(claim_type=as.factor(claim_type))%>%
```

```
        ggplot(aes(x=claim_type,fill=claim_type)) +
        geom_bar()+
        scale_fill_manual(values=c("royalblue", "#a71429"),
        labels=c("Claim type 0","Claim type 1"))+
        guides(fill = guide_legend(title="Claim type")) +
        theme_bw()+
        labs(title="",
        x="Claim Type",
        y="")+
        theme(axis.text=element_text(size=20),
        axis.title.y = element_text(size=20),
        axis.title.x  = element_text(size=20),
        legend.text = element_text(size=20))
> p2
```

B Code for plotting feature-dependent development factors

Chain ladder

```
> dtb_2_plot_M <- resurv.fit.predict.M$hazard_frame_output
b_2_plot_M=dtb_2_plot_M %>%
      mutate(DP_o=48-DP_rev_o+1)

> dtb_2_plot_Q <- resurv.fit.predict.Q$hazard_frame_output

> dtb_2_plot_Q=dtb_2_plot_Q %>%
      mutate(DP_o=16-DP_rev_o+1)

> dtb_2_plot_Y <- resurv.fit.predict.Y$hazard_frame_output

> dtb_2_plot_Y=dtb_2_plot_Y %>%
      mutate(DP_o=4-DP_rev_o+1)
```

```
> CL = resurv.fit.cox$IndividualDataPP$training.data %>%
        mutate(DP_o =
        max(resurv.fit.predict.Q$hazard_frame_output$DP_rev_o)-DP_rev_o + 1) %>%
        group_by(AP_o, DP_o) %>%
        summarize(I=sum(I), .groups="drop") %>%
        group_by(AP_o) %>%
        arrange(DP_o) %>%
        mutate(I_cum = cumsum(I),
               I_cum_lag = lag(I_cum, default=0)) %>%
        ungroup() %>%
        group_by(DP_o) %>%
        reframe(df_o =
        sum(I_cum*(AP_o<=max(resurv.fit.cox$IndividualDataPP$training.data$AP_o)-DP_o+1)) /

          sum(I_cum_lag*(AP_o<=max(resurv.fit.cox$IndividualDataPP$training.data$AP_o)-DP_o+1)),
                I=sum(I*(AP_o<=max(resurv.fit.cox$IndividualDataPP$training.data$AP_o)
                -DP_o))) %>%
        mutate(DP_o_join = DP_o-1) %>%as.data.frame()

> CL %>%
        filter(DP_o>1) %>%
        ggplot(aes(x=DP_o,
                   y=df_o))+
        geom_line(linewidth=2.5,color="#454555") +
        labs(title="Chain ladder",
             x = "Development quarter",
             y = "Development factor") +
        ylim(1,max(dtb_2_plot_Q$df_o)+.01)+
        theme_bw(base_size=rel(5))+
```

```
        theme(plot.title = element_text(size=20))
```

```
> CL_months = individual_dataM$training.data %>%
        mutate(DP_o = max(resurv.fit.predict.M$hazard_frame_output$DP_rev_o)-DP_rev_o + 1) %>%
        group_by(AP_o, DP_o) %>%
        summarize(I=sum(I), .groups="drop") %>%
        group_by(AP_o) %>%
        arrange(DP_o) %>%
        mutate(I_cum = cumsum(I),
               I_cum_lag = lag(I_cum, default=0)) %>%
        ungroup() %>%
        group_by(DP_o) %>%
        reframe(df_o = sum(I_cum*(AP_o<=max(individual_dataM$training.data$AP_o)-DP_o+1)) /
                  sum(I_cum_lag*(AP_o<=max(individual_dataM$training.data$AP_o)-DP_o+1)),
                I=sum(I*(AP_o<=max(individual_dataM$training.data$AP_o)-DP_o))) %>%
        mutate(DP_o_join = DP_o-1) %>%as.data.frame()
> ticks.at <- seq(1,48,4)
> labels.as <- as.character(ticks.at)

> CL_months %>%
        filter(DP_o>1) %>%
        ggplot(aes(x=DP_o,
                   y=df_o))+
        geom_line(linewidth=2.5,color="#454555") +
        labs(title="Chain ladder",
             x = "Development month",
             y = "Development factor") +
        ylim(1, 2.5+.01)+
        scale_x_continuous(breaks = ticks.at,
                           labels = labels.as) +
```

```
        theme_bw(base_size=rel(5))+
        theme(plot.title = element_text(size=20))


> CL_years = individual_dataY$training.data %>%
        mutate(DP_o = max(resurv.fit.predict.Y$hazard_frame_output$DP_rev_o)-DP_rev_o + 1) %>%
        group_by(AP_o, DP_o) %>%
        summarize(I=sum(I), .groups="drop") %>%
        group_by(AP_o) %>%
        arrange(DP_o) %>%
        mutate(I_cum = cumsum(I),
               I_cum_lag = lag(I_cum, default=0)) %>%
        ungroup() %>%
        group_by(DP_o) %>%
        reframe(df_o = sum(I_cum*(AP_o<=max(individual_dataM$training.data$AP_o)-DP_o+1)) /
                  sum(I_cum_lag*(AP_o<=max(individual_dataM$training.data$AP_o)-DP_o+1)),
                I=sum(I*(AP_o<=max(individual_dataM$training.data$AP_o)-DP_o))) %>%
        mutate(DP_o_join = DP_o-1) %>%as.data.frame()
> ticks.at <- seq(1,4,1)
> labels.as <- as.character(ticks.at)

> CL_years %>%
        filter(DP_o>1) %>%
        ggplot(aes(x=DP_o,
                   y=df_o))+
        geom_line(linewidth=2.5,color="#454555") +
        labs(title="Chain ladder",
             x = "Development year",
             y = "Development factor") +
        ylim(1, 2.5+.01)+
        scale_x_continuous(breaks = ticks.at,
                           labels = labels.as) +
```

```
theme_bw(base_size=rel(5))+
theme(plot.title = element_text(size=20))
```

Quarterly Output

```
> dtb_2_plot_M <- resurv.fit.predict.M$hazard_frame_output
> dtb_2_plot_M=dtb_2_plot_M %>%
  mutate(DP_o=48-DP_rev_o+1)

> dtb_2_plot_Q <- resurv.fit.predict.Q$hazard_frame_output

> dtb_2_plot_Q=dtb_2_plot_Q %>%
  mutate(DP_o=16-DP_rev_o+1)

> dtb_2_plot_Y <- resurv.fit.predict.Y$hazard_frame_output

> dtb_2_plot_Y=dtb_2_plot_Y %>%
    mutate(DP_o=4-DP_rev_o+1)

  ticks.at <- seq(1,16,by=2)
  labels.as <- as.character(ticks.at)

> ap=15
> ct=1
> dtb_2_plot_Q %>%
  filter(claim_type==ct,
          AP_o==ap,
          DP_o>1) %>%
          ggplot(aes(x=DP_o,
           y=df_o))+
           geom_line(linewidth=2.5,color="royalblue") +
           ylim(1,max(dtb_2_plot_Q$df_o)+.01)+
```

```
          labs(title=paste("COX: Accident Quarter", ap, "Claim Type", ct),
          x = "Development quarter",
          y = "Development factor") +
          scale_x_continuous(breaks = ticks.at,
          labels = labels.as) +
          theme_bw(base_size=rel(5))+
          theme(plot.title = element_text(size=20))
```

```
> ap=12
> ct=0
> dtb_2_plot_Q %>%
lter(claim_type==ct,
          AP_o==ap,
          DP_o>1) %>%
          ggplot(aes(x=DP_o,
           y=df_o))+
          geom_line(linewidth=2.5,color="royalblue") +
          ylim(1,max(dtb_2_plot_Q$df_o)+.01)+
          labs(title=paste("COX: Accident Quarter", ap, "Claim Type", ct),
          x = "Development quarter",
          y = "Development factor") +
          scale_x_continuous(breaks = ticks.at,
                   labels = labels.as) +
                   theme_bw(base_size=rel(5))+
                   theme(plot.title = element_text(size=20))
```

Monthly Output

```
> ct=0
> ap=36
> dtb_2_plot_M %>%
```

```
lter(claim_type==ct,
           AP_o==ap,
           DP_o>1) %>%
           ggplot(aes(x=DP_o,
               y=df_o))+
               geom_line(linewidth=2.5,color="#a71429") +
               ylim(1,2.5+.01)+
               labs(title=paste("XGB: Accident Month", ap, "Claim Type", ct),
         x = "Development month",
         y = "Development factor") +
         scale_x_continuous(breaks = ticks.at,
                       labels = labels.as) +
                       theme_bw(base_size=rel(5))+
                       theme(plot.title = element_text(size=20))


> ct=1
> ap=7
> dtb_2_plot_M %>%
            filter(claim_type==ct,
                   AP_o==ap,
                   DP_o>1) %>%
            ggplot(aes(x=DP_o,
                       y=df_o))+
            geom_line(linewidth=2.5,color="#a71429") +
            ylim(1,max(dtb_2_plot_M$df_o)+.01)+
            labs(title=paste("COX: Accident Month", ap, "Claim Type", ct),
                 x = "Development month",
                 y = "Development factor") +
            scale_x_continuous(breaks = ticks.at,
                               labels = labels.as) +
            theme_bw(base_size=rel(5))+
```

```
        theme(plot.title = element_text(size=20))
```

Yearly Output

```
> ct=0
> ap=2
> dtb_2_plot_Y %>%
    filter(claim_type==ct,
           AP_o==ap,
           DP_o>1) %>%
    ggplot(aes(x=DP_o,
               y=df_o))+
    geom_line(linewidth=2.5,color="#FF6A7A") +
    ylim(1,2.5+.01)+
    labs(title=paste("COX: Accident Year", ap, "Claim Type", ct),
         x = "Development year",
         y = "Development factor") +
    scale_x_continuous(breaks = ticks.at,
                       labels = labels.as) +
    theme_bw(base_size=rel(5))+
    theme(plot.title = element_text(size=20))


> ct=1
> ap=3

> dtb_2_plot_Y %>%
    filter(claim_type==ct,
           AP_o==ap,
           DP_o>1) %>%
    ggplot(aes(x=DP_o,
               y=df_o))+
    geom_line(linewidth=2.5,color="#FF6A7A") +
```

```
ylim(1,2.5+.01)+
labs(title=paste("COX: Accident Year", ap, "Claim Type", ct),
     x = "Development year",
     y = "Development factor") +
scale_x_continuous(breaks = ticks.at,
                   labels = labels.as) +
theme_bw(base_size=rel(5))+
theme(plot.title = element_text(size=20))
```

C Code for computing $\mathtt{ARE}^{\mathtt{TOT}}$ and $\mathtt{ARE}^{\mathtt{CAL}}$

The computations in this Appendix rely on the `dplyr` and `tidyr` packages from the `tidyverse` (Wickham et al., 2019).

```
> library(dplyr)
> library(tidyr)
```

Computation of $\mathtt{ARE}^{\mathtt{TOT}}$

```
> true_output <-resurv.fit.predict.Q$ReSurvFit$IndividualData$full.data%>%
    mutate(
    DP_rev_o = floor(max_dp_i*conversion_factor)-
    ceiling(DP_i*conversion_factor+((AP_i-1)%%(1/conversion_factor))*
    conversion_factor) +1,
    AP_o = ceiling(AP_i*conversion_factor),
    TR_o= AP_o-1
    ) %>%
    filter(DP_rev_o <=TR_o) %>%
    group_by(claim_type, AP_o, DP_rev_o) %>%
    mutate(claim_type = as.character(claim_type)) %>%
    summarize(I=sum(I), .groups = "drop") %>%
    filter(DP_rev_o>0)
```

```
#Total output
> score_total<-
surv.fit.predict.Q$hazard_frame_output[,
"claim_type","AP_o", "DP_rev_o", "I_expected")] %>%
          inner_join(true_output, by =c("claim_type","AP_o", "DP_rev_o")) %>%
          mutate(ave = I-I_expected,
          abs_ave = abs(ave)) %>%
          ungroup()

> are_tot=abs(sum(score_total$ave))/sum(score_total$I)
> are_tot
  0.1161133
```

Computation of $\mathtt{ARE}^{\mathtt{CAL}}$

```
> dfs_output <- resurv.fit.predict.Q$hazard_frame_output %>%
  select(AP_o, claim_type, DP_rev_o, df_o) %>%
  mutate(DP_rev_o = DP_rev_o) %>%
  distinct()

> score_diagonal <-
surv.fit.predict.Q$ReSurvFit$IndividualData$full.data  %>%
          mutate(
          DP_rev_o = floor(max_dp_i*conversion_factor)-
          ceiling(DP_i*conversion_factor+((AP_i-1)%%(1/conversion_factor))*
          conversion_factor) +1,
          AP_o = ceiling(AP_i*conversion_factor)) %>%
          group_by(claim_type, AP_o, DP_rev_o) %>%
          mutate(claim_type = as.character(claim_type)) %>%
          summarize(I=sum(I), .groups = "drop") %>%
          group_by(claim_type, AP_o) %>%
          arrange(desc(DP_rev_o)) %>%
```

```
        mutate(I_cum=cumsum(I)) %>%
        mutate(I_cum_lag = lag(I_cum, default = 0)) %>%
        inner_join(dfs_output, by = c("AP_o", "claim_type", "DP_rev_o")) %>%
        mutate(ave_2 =  I_cum-I_cum_lag * df_o,
        abs_ave_2 = abs(ave_2),
        RP_o = max(DP_rev_o)-DP_rev_o + AP_o) %>%
        inner_join(true_output[,c("AP_o", "DP_rev_o")] %>%
        distinct(), by =c("AP_o", "DP_rev_o")) %>%
        group_by(RP_o) %>%
        reframe(ave_2 = sum(ave_2),
        I=sum(I))

> are_cal_q = sum(abs(score_diagonal$ave_2)*score_diagonal$I)/sum(score_diagonal$I)
> are_cal_q
  0.1065027
```