baggingReserve: Bagging and Regression Trees for Individual Claims Reserving in R

Jan Janoušek
Charles University

Michal Pešta
Charles University

Abstract

The R package **baggingReserve** provides three things for claim-level loss reserving: a data contract that turns a table of claims and a table of payment transactions into a claim-by-development-year panel at a valuation date; a recursion engine that completes the payment history of every reported claim with one regression per development year, through an interface for user-supplied learners and with plugged-in or simulated completion; and back-tests that score a prediction, by calendar year and separately for reported claims and claims incurred but not reported (IBNR), against realised payments and a reference method such as the chain ladder. The default baseline is the method of Janoušek and Pešta (2025): regression trees with bagging, whose replicates describe resampling variation, not a predictive distribution, and chain-ladder projections of the IBNR claim counts. On a shipped synthetic portfolio, further portfolios from the same generator and an illustrative motor third-party liability dataset, the standard chain ladder is more accurate than the bagged trees on most portfolios. A correctly specified simulation shows that the default plug-in completion is biased for multi-year totals when whether a claim pays depends on its payment history; simulated completion removes most of that bias. Only static claim features are supported. The package makes such checks routine, so that claim-level methods can be tested on a user's own data before they are relied upon.

Keywords: claim-level reserving, back-testing, bagging, regression trees, chain ladder, R.

This vignette is the manuscript submitted to the Journal of Statistical Software*, compiled with* ***baggingReserve*** *0.2.0, the version submitted with it. The supplementary material, the replication scripts and the deposited results accompany the submission.*

1. Introduction

A non-life insurer must hold reserves for the payments it still owes on claims that have already occurred. At a valuation date t, these claims are of two kinds: claims already reported to the insurer, open or closed (a closed claim can reopen), and claims incurred but not yet reported (IBNR). Classical reserving methods such as the chain ladder (Mack 1993; England and Verrall 2002) work on run-off triangles, i.e., on payments aggregated by accident year and development year; machine-learning extensions of the aggregate models exist as well (Gabrielli, Richman, and Wüthrich 2020). Aggregation discards the information carried by the individual claim: its type, characteristics of the insured, reporting time and payment history. Individual (micro-level) reserving methods keep it; see Wüthrich (2018) for a machine-learning formulation on which the method implemented here builds. Tree-based individual

methods include the censored regression trees of Lopez, Milhaud, and Thérond (2016, 2019), the nonparametric approach of Baudry and Robert (2019) and the gradient boosting of Duval and Pigeon (2019); Delong, Lindholm, and Wüthrich (2022) use individual claims data in a collective reserving model, and Hiabu, Hofman, and Pittarello (2023) forecast the numbers of IBNR claims from individual reporting data with feature-dependent development factors.

In what we call the source article, Janoušek and Pešta (2025) predict the outstanding payments claim by claim. For each development year, one regression tree predicts a reported claim's payment in the next year from its static features and its payment history; the trees are applied recursively to complete every reported claim's payment history; bootstrap aggregating (bagging, Breiman 1996) yields replicate reserve predictions and an out-of-bag (OOB) diagnostic; and payments of IBNR claims are estimated from chain-ladder projections of the reporting counts and the average payment by development year. The method was originally implemented in a script around **rpart** (Therneau and Atkinson 2026) and **ipred** (Peters and Hothorn 2026).

This paper presents **baggingReserve**, an R package for claim-level reserving and for its evaluation against realised payments. It makes three contributions. First, a *data contract*: the constructor `claims_panel()` takes a table of claims and a table of payment transactions with user-specified column names, enforces the valuation date on both, distinguishes observed zeros from future cells, keeps recoveries and claims without payments, and rejects input it cannot interpret. Second, a *recursion engine* shared by the single-tree fit, the bagged ensemble and user-supplied learners, with a checked learner interface and two completion modes, plugged-in means and simulated payments; its bagging draws random numbers from streams allocated to replicates and development years, so that the results do not depend on the number of worker processes, and each prediction states what its columns and their quantiles mean. Third, *backtesting*: `backtest()` compares a prediction with the payments that followed, by calendar year and separately for reported and IBNR claims, and scores a reference method such as the chain ladder against the same truth; valuing a portfolio at an earlier date turns any claims history into such a test.

The method of the source article is the default estimator, a documented, reproducible claim-level baseline against which other learners and the chain ladder can be measured; it is not more accurate than the chain ladder on most portfolios studied here (Section 4.5), and it has not been validated against the source article's numbers, whose portfolio is not available. The package is best understood as a harness for finding out whether claim-level information helps on one's own data.

Table 1 compares the package with related software. **ChainLadder** (Gesmann, Murphy, Zhang, Carrato, Wüthrich, Concina, and Dal Moro 2026) in R and **chainladder** (**chainladder** Developers 2026) in Python implement aggregate reserving on triangles, from the chain ladder with Mack's standard errors to bootstrap and further models; **baggingReserve** uses only the standard chain ladder, as a comparator and for the count projection, and is no substitute for these packages in aggregate reserving. The double chain ladder of Martínez Miranda, Nielsen, and Verrall (2012), implemented in **DCL** (Martínez-Miranda, Nielsen, and Verrall 2022), adds the triangle of reported claim counts to the paid triangle, estimates a reporting delay, a payment delay and a mean payment per claim, and so separates the reserve into reported but not settled (RBNS) and IBNR claims, with a bootstrap for its predictive distribution. Its IBNR part has the structure of the IBNR component here, projected counts times a payment per claim, and it is the natural stronger aggregate comparator; this paper does not use it (Section 3.4).

Software	Input	Target	IBNR claims	Open vs. closed claims	Beyond the point prediction
ChainLadder, **chainladder**	triangle	aggregate payments	in the aggregate	not separated	Mack standard errors, bootstrap, further aggregate models
DCL	triangles of counts and payments	aggregate payments, RBNS and IBNR	counts × estimated severity	not separated	bootstrap predictive distribution
ReSurv	individual claims	numbers of IBNR claims	counts only	–	none
hirem	individual claims	payments of open reported claims	not modelled	settlement layer	simulated paths
baggingReserve	claims + payments	payments per claim and development year	counts × pooled mean payment	not modelled (Section S5)	replicate quantiles (resampling variation only); OOB diagnostic; back-tests

Table 1: Software related to **baggingReserve**. Only the Mack and bootstrap outputs of the chain-ladder packages, the bootstrap of **DCL** and the simulated paths of **hirem** quantify reserve uncertainty (Section 2.3 explains the replicate quantiles and out-of-bag diagnostic of **baggingReserve**).

ReSurv (Hofman, Pittarello, and Hiabu 2026) implements the IBNR count models of Hiabu *et al.* (2023) and does not model payments; its counts could replace the chain-ladder count projection of the IBNR component, which the package does not automate. **hirem** (Crevecoeur, Robben, and Antonio 2022; Crevecoeur and Robben 2022) fits sequential layers (settlement, payment occurrence, payment size) to one row per claim and development year counted from reporting and simulates the payment paths of the claims open at the valuation date; claims not yet reported are outside the model and a closed claim pays nothing further. It thus models the payment process of open claims more richly, whereas **baggingReserve** completes every reported claim, open or closed, from the two raw tables with one regression per development year, allows payments of either sign and adds an IBNR component, but cannot allocate the reported-claim reserve reliably between open and closed claims; on that target **hirem** is the more accurate of the two (Section 4.5). Generic ensemble packages (Peters and Hothorn 2026; Liaw and Wiener 2002; Wright and Ziegler 2017) supply learners but none of the reserving structure, and **SynthETIC** (Avanzi, Taylor, Wang, and Wong 2021) and **SPLICE** (Avanzi, Taylor, and Wang 2023) simulate the individual claims data such methods are tested on.

Section 2 states the method as implemented and Section 3 describes the package. Section 4 illustrates it on a toy example, on the shipped portfolio, at earlier valuation dates and on a motor third-party liability dataset, summarises the evaluation of the method, and describes how to reproduce the results. Section 5 concludes. The supplementary material, whose sections are cited as S1 to S9, holds the detailed evaluation: the generator of the shipped data, a second portfolio with another data layout, further portfolios, the open/closed allocation, a changing business mix, studies of simulated completion, other learners and **hirem**, the motor case in detail, and the leaf size of the bootstrap trees.

2. Reserving with regression trees and bagging

2.1. Notation and data

Time is discrete, in years. A claim r has an accident year a_r, is reported with a delay of $d_r \in \{0, 1, \dots\}$ years, and generates incremental payments $X_{r,0}, X_{r,1}, \dots$, where $X_{r,l}$ is the total paid in development year l, i.e., in calendar year $a_r + l$; payments can be negative (recoveries) and are zero before the claim is reported. A maximum development year L is fixed: payments beyond L are outside the model. The valuation date is the end of year t. The insurer then knows every claim with $a_r + d_r \leq t$ (the reported claims), and for each of them the payments $X_{r,l}$ with $a_r + l \leq t$; the cells with $a_r + l > t$ represent future payments. Summing over the claims of an accident year i gives the familiar run-off triangle, observed above the diagonal $i + l = t$.

The static features of a claim (in the source article: type, age of the insured, weekday of the accident) and its reporting delay form the covariates. Let $\mathbf{x}_{r,l} = (\text{features}_r, d_r, X_{r,0}, \dots, X_{r,l})$ denote the information about claim r up to development year l. The model of the source article assumes that $X_{r,l+1}$ given $\mathbf{x}_{r,l}$ has a conditional distribution $p_l(\mathbf{x}_{r,l})$ that does not depend on the accident year; the regression tree estimates the conditional mean of that distribution. With $\mathcal{F}_t$ the information available at the valuation date, the target is the reserve $\mathrm{E}[\sum_r \sum_{l \leq L,\, a_r + l > t} X_{r,l} \mid \mathcal{F}_t]$, the sum running over reported and unreported claims; for an unreported claim, $a_r + d_r > t$, the summands are its whole payments $X_{r,l}$ with $l \geq d_r$, which Section 2.4 estimates in the aggregate. For a reported claim more than one year from the valuation date, the method plugs the predicted mean of each future payment into the history used by the next tree (Section 2.2). Because the trees are nonlinear functions of the history, the mean of a tree's prediction is in general not the tree's prediction at the mean history, so this recursion approximates the conditional expectation and does not equal it; Section 3.3 describes the simulated alternative.

2.2. One tree per development year and recursive completion

For every development year $l = 0, \dots, L-1$, a regression tree T_l is fitted with response $X_{r,l+1}$ and covariates $\mathbf{x}_{r,l}$ on the training set

$$S_l = \{r : d_r \leq l \text{ and } a_r + l + 1 \leq t\}, \tag{1}$$

the claims that were reported by development year l and whose payment in development year $l + 1$ is observed at the valuation date. Payments before reporting are structurally zero and are neither predicted nor used as responses: a claim enters the training set of development year l only from its reporting year on. The trees are **rpart** analysis-of-variance (ANOVA) trees (Therneau and Atkinson 2026; Breiman, Friedman, Olshen, and Stone 1984), grown with complexity parameter zero and pruned to the subtree with the smallest 10-fold cross-validated error (the simplest one on ties), as in the source article, not by **rpart**'s customary one-standard-error rule, which prunes more.

Prediction proceeds year by year. The tree T_l predicts $X_{r,l+1}$ for the claims with $a_r + l + 1 > t$; its payment-history covariates $X_{r,0}, \dots, X_{r,l}$ are observed where $a_r + k \leq t$ and are the predictions of earlier trees where $a_r + k > t$. Every reported claim is completed, including claims recorded as closed, because a claim can reopen; the trees therefore do not distinguish

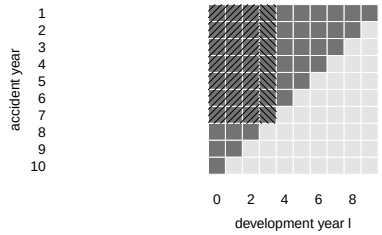


Figure 1: Cells of the run-off triangle for a portfolio with $t = 10$ and $L = 9$ (smaller than the shipped one, for legibility): rows represent accident years (each standing for all claims of that year) and columns represent development years. Dark: observed at the valuation date. Light: predicted recursively, development year by development year from left to right. Hatched: training cells for the tree indexed by $l = 2$, covariates (columns 0–2, hatched one way) and response (column 3, hatched the other way) of the claims reported by development year 2 with accident year ≤ 7.

open from closed claims, and naming the status at the valuation date as a feature does not fix this (Section S5). Figure 1 shows the pattern of observed, predicted and unavailable cells.

2.3. Bagging, replicates and the out-of-bag score

Bagging replaces each tree T_l by an ensemble $T_{l,1}, \ldots, T_{l,B}$. Each tree $T_{l,b}$ is grown on a bootstrap sample (drawn with replacement, duplicates kept) of S_l, drawn independently for each development year rather than once per claim. Unlike the single trees of Section 2.2, the bootstrap trees are grown with complexity parameter zero and left unpruned, without cross-validation (pruning and cross-validation are configurable through arguments), as in the source article's code. The source article and the package define *replicate* b as the completion of all payment histories with the trees $T_{0,b}, \ldots, T_{L-1,b}$, one per development year, applied recursively as in Section 2.2. Writing F for the completion map,

$$R_b = F(T_{0,b}, \ldots, T_{L-1,b}), \qquad \bar{R} = \frac{1}{B} \sum_{b=1}^{B} R_b, \tag{2}$$

the bagged reserve prediction is the mean $\bar{R}$ of the completed replicates. Because the recursion is nonlinear, this is not the same as completing the histories once with tree predictions averaged within each development year, which the package does not implement. The empirical quantiles of $R_1, \ldots, R_B$, the *replicate quantiles*, describe only the variation, under bootstrap resampling, of the reserve predicted by one set of unpruned bootstrap trees: the numbers of unreported claims are projected once and held fixed across replicates (Section 2.4), no payment noise is simulated, and because the samples are drawn per development year they do not reproduce how the errors of successive trees are linked through the same claims. The replicate quantiles are therefore neither prediction intervals for future payments nor confidence intervals for the bagged mean, a realised value inside their range is no coverage statement, and the package delivers no predictive distribution of the reserve ("resampling variation" below).

For the out-of-bag diagnostic of development year l, let $\mathcal{B}_l(r)$ be the set of replicates whose bootstrap sample of S_l did not contain claim r, and let $\mathrm{OOB}_l(r) = |\mathcal{B}_l(r)|^{-1} \sum_{b \in \mathcal{B}_l(r)} T_{l,b}(\mathbf{x}_{r,l})$ denote the average prediction of those trees, computed from the claim's *observed* history.

With $S_l^* = \{r \in S_l : |\mathcal{B}_l(r)| > 0\}$ denoting the set of covered claims and $y_r = X_{r,l+1}$,

$$\text{OOB-error}_l = \frac{1}{|S_l^*|} \sum_{r \in S_l^*} (y_r - \text{OOB}_l(r))^2, \qquad \text{OOB-score}_l = 1 - \frac{\sum_{r \in S_l^*} (y_r - \text{OOB}_l(r))^2}{\sum_{r \in S_l} (y_r - \bar{y}_l)^2}, \quad (3)$$

where the denominator runs over all training claims of development year l and $\bar{y}_l$ is their mean response. With complete coverage the score is interpreted like R^2: one for a perfect fit, negative when the ensemble is worse than predicting the mean. A claim is left out of a bootstrap sample of n_l claims with probability $(1 - 1/n_l)^{n_l} \approx e^{-1}$, so it is covered with probability about $1 - (1 - e^{-1})^B$: coverage is essentially complete for B in the hundreds but not for very small B, and then numerator and denominator run over different sets and the score is biased upwards. The package therefore reports the coverage and also the ordinary coefficient of determination over the covered claims alone (`oob_score_covered`). The score assesses the one-year regressions given observed histories; it does not validate the recursive multi-year forecast or the IBNR component.

2.4. Claims incurred but not reported

Let $N_{i,d}$ be the number of claims of accident year i reported with delay d; the triangle of these counts is observed for $i + d \leq t$. Theorem 1 of the source article expresses the expected payments of the unreported claims of accident year i as the product of the expected number of claims and the expected payments per claim, when the payments of a claim reported after t are independent of the information at t; the unreported claims have no recorded features, delay or history, so the trees cannot be applied to them. The package implements the source article's estimator of this product. It completes the count triangle by a chain-ladder projection, which is justified when claims arrive as a homogeneous marked Poisson process whose reporting-delay distribution does not depend on the accident year (Verrall and Wüthrich 2016, Section 6.1); $\widehat{U}_{i,l} = \sum_{d \leq l} \widehat{N}_{i,d}\, \mathbf{1}(i + d > t)$ is the projected number of claims of accident year i reported after the valuation date with delay at most l. The payment per such claim in development year l is estimated by the pooled mean payment of the reported claims in that development year,

$$\hat{\mu}_l = \frac{\sum_i \widehat{R}_{i,l}}{\sum_i \sum_{d \leq l,\ i+d \leq t} N_{i,d}}, \quad (4)$$

where, for a fixed replicate, $\widehat{R}_{i,l}$ denotes the sum of the observed and predicted payments of the reported claims of accident year i in development year l, and the denominator counts the reported claims of all accident years with delay at most l (observed counts only). Because the numerator contains the replicate's predicted payments of the reported claims, an error of the reported-claim component propagates into the IBNR component. The IBNR payment in cell (i, l) is $\widehat{U}_{i,l}\, \hat{\mu}_l$; it is zero in observed cells because every claim of accident year i reported after t has delay $d > t - i$. The projected counts are the same in every replicate, so the replicate spread of the IBNR component comes from $\hat{\mu}_l$ alone and contains no count uncertainty.

The estimator transfers the pooled payment behaviour of the reported claims to the late-reported claims. It requires that, in development year l, a claim of accident year i reported after t, with delay $t - i < d \leq l$, has the same expected payment as the pooled mean of the reported claims with delay at most l. Because $\hat{\mu}_l$ pools all accident years, this needs both

Function	Returns	Methods and functions applying to the result
`claims_panel()`	'claims_panel'	`print`, `summary`, `as.data.frame`, `as_triangle()`, `claim_counts()`
`reserve_tree()`	'reserve_tree'	`print`, `summary`, `predict`, `plot`
`bag_reserve()`	'bag_reserve'	`print`, `summary`, `predict`, `plot`, `oob_score()`
`predict()` on a fit	'reserve_prediction'	`print`, `summary`, `plot`, `as.data.frame`, `as_triangle()`, `calendar_year_payments()`, `claim_reserves()`
`ibnr_reserve()`	'reserve_prediction'	adds the IBNR component to a prediction
`chain_ladder()`	'chain_ladder'	`print`, `as_triangle()`, `calendar_year_payments()`
`backtest()`	'reserve_backtest'	`print`, `as.data.frame`
`two_part_learner()`	list	the argument `learner` of `reserve_tree()` and `bag_reserve()` (Section 3.3)

Table 2: Exported functions, the classes of their return values and the methods and functions that apply to them. Objects of these classes are S3 lists; `as_triangle()`, `calendar_year_payments()` and `oob_score()` are generics.

that, given l, the expected payment does not depend on the reporting delay among claims with $d \leq l$, and that it does not depend on the accident year: a trend in the payments across accident years, such as the inflation of the generator of Section 4.2, also biases $\hat{\mu}_l$ for the recent accident years that carry most IBNR claims. A payment pattern that shifts with the delay, $g(l-d)$ say, violates this condition even when every claim has the same payment pattern after reporting: with delays 0 and 1 and n_0 and n_1 reported claims, respectively, the pooled mean differs from the expected payment of a late-reported claim with delay 1 by $n_0\,[g(l) - g(l-1)]/(n_0 + n_1)$, so a decreasing pattern makes the estimator underpredict the payments of late-reported claims. A different business mix among early- and late-reported claims also violates the condition, in either direction. The pooling assumption is the source article's working approximation; its suitability should be questioned when a portfolio's late-reported claims differ in kind (Section S3). The default count projection is the standard chain ladder on cumulative counts without rounding; the source article's incremental ratios with rounding are an option (Section 3.4).

3. Package design

The package is organised along the workflow construct → fit → predict → aggregate → diagnose → back-test. Table 2 lists the objects and their methods.

3.1. The data contract

`claims_panel()` takes two data frames, one with a row per claim and the other with a row per payment transaction, the valuation date `t`, the horizon `L` and the names of the columns holding the identifier (`id`), the accident year (`acc_year`), the reporting year or delay (`rep_year`, `rep_delay`), the payment year (`event_year`) and the amount (`paid`), with the defaults of the shipped data (Table 3 lists the other arguments). The argument `id` may name a different identifier column for each table. The constructor returns a claims table (identifier, accident year, reporting delay and features), a payment matrix with one row per reported claim and columns `X0`, ..., `XL`, the observation mask and construction diagnostics. Claims

are retained if reported by `t`, and payments are summed by claim and development year; times must be integers (dates are converted by the user, as in Section 4.4), on any regular grid: years here, but quarters or months numbered consecutively work alike. Observed cells without payments contain `0`, cells after the valuation date `NA`; recoveries and claims without transactions are retained. The constructor rejects duplicated or missing identifiers, payments for claims absent from the claims table or for claims not reported by `t`, payments dated before the accident or the reporting year, reporting delays or observed payments beyond `L` unless `beyond_L = "drop"` sets them aside (for a long history), and payments dated after `t` unless `future_payments = "drop"` does so; the numbers and amounts set aside are recorded, and the identifiers of the claims set aside travel with the predictions to `backtest()`. Features are never selected automatically, and only static features are supported: information that changes over the life of a claim, such as its settlement status or case reserve at the end of each development year, cannot enter the panel. This is the main limitation of the design and the reason why the package cannot allocate the reserve between open and closed claims. Character columns become factors with sorted levels; missing feature values are rejected unless `na_features = "surrogate"` lets **rpart**'s surrogate splits handle them. The reporting delay is always a covariate, and `L` is never inferred from the data. `as_triangle()` sums the panel into the paid triangle (incremental or cumulative) or the triangle of reporting counts, with `NA` in the cells after the valuation date.

3.2. Fitting and prediction

`reserve_tree()` fits the L pruned trees of Section 2.2; `bag_reserve()` fits the $B \times L$ bootstrap trees of Section 2.3. In the package's arguments, printed output and tables, the development-year index l of a tree is called its `period`. Both functions use the same internal recursion engine. For each development year, the eligibility conditions determine the training rows and the model frame; the frame's column order (features, reporting delay, `X0`, ..., `Xl`, response) is fixed, so that **rpart**'s tie-breaking between equally good splits is reproducible; and the recursive completion overwrites only the future cells of the current step, using the completed cells of the earlier steps. All fits in this paper use the package's default `rpart.control()` settings: `cp = 0`, `xval = 10`, `minsplit = 20`, `minbucket = 7`, `maxdepth = 30` for the single trees, which are then pruned to the subtree with the smallest cross-validated error (the simplest one on ties), and `cp = 0`, `xval = 0`, `minsplit = 20`, `minbucket = 7`, `maxdepth = 30` for the unpruned bootstrap trees. No hyperparameter was tuned on any of the portfolios, and the out-of-bag scores were not used to select any setting.

Calling `predict()` on either fit returns a '`reserve_prediction`': the observed incremental triangle, a table of the future cells (accident year, development year, calendar year), and a matrix with one row per future cell and one column per replicate of the predicted payments of the reported claims (one column for a single-tree fit), and the completed claim-level payment matrix, from which `claim_reserves()` returns the reserve of every reported claim. `calendar_year_payments()` sums the cells along the diagonals into payments per future calendar year plus their total through development year L; `summary()` gives the mean, standard deviation and empirical quantiles of these across the replicates, with the quantile type as an argument. The default is `type = 7`, R's default; the source article used order statistics, which `type = 1` reproduces.

3.3. Other learners and simulated completion

Both fitting functions accept an argument `learner`, which replaces the **rpart** trees in every development year while the eligibility rules, the bootstrap, the recursion, the out-of-bag score and the random-number streams stay the package's. A learner is a list of functions: `fit(formula, data)` returns a model for one development year l; `formula` is `X`$(l+1)$ `~ .` and `data` the model frame of Section 3.2, with the features, `rep_delay` and the payment history `X0`, ..., `X`l as columns, one row per training claim (for `bag_reserve()`, per bootstrap draw). `predict(model, newdata)` returns one number per row of `newdata`, which has the same columns without the response. An optional `simulate(model, newdata)` returns one simulated payment per row. Every prediction of a learner, for the completion and the out-of-bag votes alike, receives this frame and is checked: a numeric vector with one finite value per row, else an error naming the stage and the development year. `two_part_learner()` builds a learner from one model for whether a claim pays in the development year, fitted to a 0/1 response, and one for the amount of a non-zero payment, and checks the output of each, probabilities in $[0, 1]$ included, before combining them; it predicts $p \cdot s$ and simulates by drawing the payment indicator, the amount being the predicted s unless the size model has a `simulate()` of its own. Here both parts are regression trees, on the claims of type 3 of the shipped portfolio:

```
R> library("baggingReserve")
R> data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
R> cl3 <- synthetic_claims[synthetic_claims$Type == "3", ]
R> pay3 <- synthetic_payments[synthetic_payments$Id %in% cl3$Id, ]
R> panel3 <- claims_panel(cl3, pay3, t = 12, L = 11,
+    features = c("Age", "AccWeekday"))
R> tree_learner <- function(...) list(
+    fit = function(formula, data) rpart::rpart(formula, data = data,
+      method = "anova", control = rpart::rpart.control(...)),
+    predict = function(model, newdata) unname(predict(model, newdata)))
R> tp <- two_part_learner(occurrence = tree_learner(cp = 0.001, minbucket = 50),
+    size = tree_learner(cp = 0.001, minbucket = 50))
R> fit_tp <- reserve_tree(panel3, seed = 1, learner = tp)
R> pred_tp <- predict(fit_tp, completion = "draw", R = 50, seed = 1)
R> cy_tp <- calendar_year_payments(pred_tp, "reported")
R> round(c(mean = mean(cy_tp["total", ]),
+    quantile(cy_tp["total", ], c(0.025, 0.975))))

   mean    2.5%   97.5%
9447840 8838521 9984443
```

Instead of plugging the predicted mean of each development year into the next, `completion = "draw"` simulates the payments: for the default trees each one is drawn from the training responses of the claim's leaf, for a user-supplied learner from its `simulate()`. The R columns of the result are simulated completions of the fitted model, so their spread is process variability given that model, without estimation variability. `bag_reserve(completion = "draw")` simulates one completion per bootstrap replicate from that replicate's trees: the spread then

adds process variability to the resampling variation of one set of unpruned bootstrap trees (still not a predictive distribution; Section 2.3). For a bagged fit the completion is chosen when fitting; `predict()` refuses a different one, and a fit with simulated completion predicts its own portfolio only; `print()`, `summary()`, `plot()` and `backtest()` describe the sources of variability of every prediction. The plug-in default reproduces the source article, but it is not a consistent estimator of the multi-year reserve when the occurrence of a payment depends on the payment history: in the correctly specified simulation of Section 4.6 its bias for the total does not vanish as the portfolio grows, while simulated completion removes most of it. For totals beyond next year we therefore recommend `completion = "draw"`, and `print()` of a plug-in prediction says so. Neither completion is the more accurate on every portfolio of Section 4 (Section S7), and the calibration of the simulated spread is not examined: a draw resamples the training responses of one leaf, few in the late development years. Section S7 applies one- and two-part gradient boosting learners built the same way.

3.4. The IBNR component and the comparators

`ibnr_reserve()` adds to a prediction a second matrix of the same shape with the IBNR payments of Section 2.4 and keeps the projected counts, the denominator of Equation (4) and the per-replicate $\hat{\mu}_l$ for inspection; counts that disagree with the triangle stored in the prediction (another portfolio, cumulative counts) are an error. `chain_ladder()` projects a triangle with `NA` in the unobserved cells by the standard chain ladder: volume-weighted development factors from the cumulative triangle over the accident years observed in both columns, projection from the latest observed value, no tail factor, point predictions only. The package's tests check its factors, completed triangle and outstanding amounts against `ChainLadder::MackChainLadder()` with unit weights. Setting the argument `projection` to `"incremental_ratio"` applies ratios of incremental column sums to the incremental cells instead, which is what the source article's code did for its chain-ladder benchmark and for the count projection; `ibnr_reserve()` provides the same option together with `round = TRUE` for the source article's rounding of the projected counts.

As a comparator, this chain ladder is deliberately the plainest aggregate method: the paid triangle alone, no tail factor, no incurred or count information. It has the same target as the claim-level method, the payments through development year L. The comparisons in this paper are with this standard chain ladder, not with the best aggregate method; a stronger one, from **ChainLadder** or the double chain ladder of **DCL** say, can be scored against the same truth through the argument `reference` of `backtest()`, which takes the prediction of any method as payments by future calendar year.

3.5. Back-testing

`backtest()` takes a prediction and the claims and payments tables of the same portfolio, including the claims reported after the valuation date, and builds the realised payments of every future cell through development year L, split into claims reported by t and claims reported later (the IBNR truth); payments beyond L are reported as the excluded tail. For each component it returns, per future calendar year and in total, the realised and predicted payments, their difference and relative error (the difference divided by the realised amount, so that its sign reverses in a calendar year of net recoveries), and the replicate range with whether the truth lies inside it (descriptive only; Section 2.3). With `reference`, a '`chain_ladder`' object

or a named vector of payments by calendar year, another method is scored against the truth of the population it predicts: all claims by default, which requires the prediction's IBNR component, or the reported claims (`reference_component = "reported"`). A '`chain_ladder`' reference must have been fitted on the same panel (accident years, development years, valuation date and observed payments are checked). The truth tables must describe the whole portfolio: the function rejects claims absent from the prediction that were reported by t, as a selection by the reported claims' identifiers would produce, and predicted claims whose accident or reporting year contradicts the prediction, which stores both; it refuses a payment before its claim's reporting year, as the constructor does, and counts the claims set aside by `beyond_L = "drop"` in the excluded tail, not as missing. Its column names default to those given to `claims_panel()`, and it takes the reporting year by the constructor's rule, from the reporting delay when the panel was built from one; without a reporting year these checks are skipped, with a warning. A future cell without transactions counts as a realised zero; when the extract ends earlier, `observed_to` names the last fully observed calendar year, and later cells are left unscored (`NA`) and out of the totals. With the reporting year, the result also compares the projected with the realised numbers of IBNR claims by accident year (component `counts`), which separates the count error of the IBNR component from the error of its mean payment.

3.6. Reproducibility and parallel execution

`bag_reserve()` processes the replicates without retaining their trees by default: a worker fits the L trees of a replicate on its bootstrap samples, records the out-of-bag predictions ("votes"), completes the panel, keeps the aggregated future cells and discards the trees, unless `keep_trees = TRUE`; only with the trees (and plug-in completion) can `predict()` complete another '`claims_panel`' with the same features, factor levels and horizon.

Its random numbers come from the `"L'Ecuyer-CMRG"` generator (L'Ecuyer, Simard, Chen, and Kelton 2002). Replicate b receives stream b after the seed, and development year l receives substream $l+1$ within that stream; a worker sets that state before drawing a bootstrap sample. Under the same software, the completed replicates are therefore identical for any number of workers, any batching of replicates and any scheduling, and the first B replicates of a larger run are those of a run with B replicates; the out-of-bag sums are accumulated batch by batch and can differ in their last floating-point digits. An explicit `seed` leaves the caller's generator kind and state unchanged; with `seed = NULL` the seed is drawn from it. `reserve_tree()` draws random numbers only for the cross-validation folds; with `seed`, it uses the generator kind `"Mersenne-Twister"` whatever the caller's kind, so the pruned trees do not depend on the session's generator. The source article's code drew its bootstrap samples with `bagging()` from **ipred** under a different seeding scheme, so its replicates cannot be reproduced exactly. Parallel execution uses forked processes on Unix (**parallel**) and a socket cluster elsewhere. Socket workers are fresh R sessions: they receive the variables of the global environment and the attached packages that the functions of a user-supplied learner refer to (found with **codetools**), so that a learner written at top level gives the same replicates on every backend; one that reaches an object only by name, through `get()` say, fails on them with a message saying so.

3.7. Practical use

The main choices are the completion, B, L, `keep_trees` and the number of workers (Table 3 lists the key arguments and defaults). The default `completion = "mean"` reproduces the source article and is kept for that reason; for totals beyond next year use `completion = "draw"`, as in the call `bag_reserve(panel, B, seed, completion = "draw")` of Section 4.2, and back-test the result at several valuation dates (Section 4.3).

The number of replicates. The Monte Carlo error of the bagged mean is the replicate standard deviation divided by $\sqrt{B}$. Over 10 bagging seeds with $B = 1000$ on the shipped portfolio, the bagged means vary with a standard deviation of 0.06% of their mean for next year and 0.34% for the total through development year 11, of the order of the replicate standard deviation divided by $\sqrt{B}$: 0.05% and 0.29% for the replicate standard deviation averaged over the 10 seeds (0.06% and 0.30% for that of the live run alone, Section S2). Replicates in the hundreds therefore fix the bagged mean far more precisely than the back-tests of Section 4.5 measure its error, and the default $B = 100$ suffices for it; the extreme replicate quantiles need more (Section S2 compares $B = 1000$ with $B = 10{,}000$), and the out-of-bag coverage needs B well above ten (it is about 99.0% at $B = 10$).

The horizon. `L` is never inferred. The last tree needs training claims with $a_r + L \leq t$, so L is at most the number of accident years less one when the latest accident year is t; `backtest()` reports the payments beyond L as the excluded tail.

Memory. On the shipped portfolio, a straightforward implementation that fits `bagging()` from **ipred** for each development year (`keepX = FALSE`, `coob = FALSE`, the same `rpart.control()`) and keeps its trees, with the components **rpart** stores by default, holds 5.1 GB for $B = 50$ replicates, against 19 MB for `bag_reserve()` with its default `keep_trees = FALSE` (`replication/engine_benchmark.R`); on one worker it took 166 s, `bag_reserve()` 247 s, which also computes the out-of-bag diagnostic and the claim-level reserve. With `keep_trees = TRUE` the package keeps all $B \times L$ unpruned trees as well, and its memory then grows linearly in B in the same way.

Workers. `ncores` sets the number of worker processes, at most the number of cores detected (a larger request is reduced, with a message; it is used as given when that number cannot be detected); `backend = "auto"` forks on Unix, where the workers share the panel, and starts a socket cluster elsewhere, where each receives a copy. The results do not depend on these choices (Section 3.6). In the timing study of Section S2 the time per replicate falls nearly in proportion to the number of workers, 7.3 times faster on 8 workers than on one; the other deposited fits ran under other loads.

4. Illustration

4.1. The data contract on a toy portfolio

Five claims and six transactions illustrate the rules. Claim `d` is reported after the valuation date $t = 3$ and is not part of the panel; claim `a` has two payments in year 1 and a recovery in year 2; claim `b` is reported one year after its accident, so its cell for development year 0 is an observed zero; claim `e` is reported but has no payment yet; the cell for development year 2 of claim `c` lies in the future.

```
R> library("baggingReserve")
```

Function	Key arguments and their defaults
`claims_panel()`	`t`, `L`, `features = character()`, `accident_years = NULL`, `na_features = c("fail", "surrogate")`, `future_payments = c("error", "drop")`, `beyond_L = c("error", "drop")`
`reserve_tree()`	`control = rpart::rpart.control(cp = 0, xval = 10)`, `prune = c("min_xerror", "none")`, `seed = NULL`, `learner = NULL`
`bag_reserve()`	`B = 100`, `seed = NULL`, `ncores = 1`, `control = rpart::rpart.control(cp = 0, xval = 0)`, `prune = c("none", "min_xerror")`, `oob = TRUE`, `keep_trees = FALSE`, `batch_size = NULL`, `backend = c("auto", "fork", "psock", "serial")`, `claim_level = TRUE`, `learner = NULL`, `completion = c("mean", "draw")`
`predict()` on a 'reserve_tree'	`newdata = NULL`, `type = c("replicates", "mean")`, `completion = c("mean", "draw")`, `R = 100`, `seed = NULL`
`ibnr_reserve()`	`counts = NULL`, `projection = c("chain_ladder", "incremental_ratio")`, `round = FALSE`
`summary()` on a 'reserve_prediction'	`probs = c(0.025, 0.25, 0.5, 0.75, 0.975, 0.995)`, `type = 7`
`backtest()`	`probs = c(0.025, 0.975)`, `type = 7`, `rep_year = "RepYear"`, `reference = NULL`, `reference_component = c("total", "reported")`, `observed_to = NULL`
`chain_ladder()`	`input = NULL`, `projection = c("chain_ladder", "incremental_ratio")`
`two_part_learner()`	`occurrence`, `size`, `min_nonzero = 20`

Table 3: Key arguments of the main functions with their defaults, read from **baggingReserve** 0.2.0 (an argument without a default must be supplied; for a vector of choices, the first is the default). `predict()` on a '`bag_reserve`' takes only `newdata` and `type`: its completion is that of `bag_reserve()`. The column names of `backtest()`, `rep_year` among them, default to those given to the prediction's `claims_panel()`.

```
R> claims <- data.frame(Id = c("a", "b", "c", "d", "e"),
+    AccYear = c(1, 1, 2, 3, 3), RepYear = c(1, 2, 2, 4, 3),
+    Type = c("x", "y", "x", "y", "x"))
R> payments <- data.frame(Id = c("a", "a", "a", "b", "c", "d"),
+    EventYear = c(1, 1, 2, 2, 3, 4), Paid = c(10, 5, -2, 7, 3, 1))
R> panel <- claims_panel(claims, payments, t = 3, L = 2, features = "Type")
R> panel

Claims panel: 4 claims reported by t = 3
  accident years 1-3 (I = 3), development years 0-2
  features: Type (+ reporting delay)
  observed cells: 9 of 12; recoveries: 1; claims without payment: 1
  dropped: 1 claim(s) not reported by t, with 1 payment(s) after t;
           0 payment(s) after t of the reported claims

R> cbind(panel$claims, panel$payments)

  id acc_year rep_delay Type X0 X1 X2
1  a        1         0    x 15 -2  0
2  b        1         1    y  0  7  0
3  c        2         0    x  0  3 NA
4  e        3         0    x  0 NA NA
```

```
R> as_triangle(panel)

Incremental payments (rows: accident years, columns: development years)
NA: not observable at t = 3
   0  1  2
1 15  5  0
2  0  3 NA
3  0 NA NA
```

A payment after the valuation date triggers an error unless it is dropped explicitly:

```
R> late <- rbind(payments, data.frame(Id = "a", EventYear = 4, Paid = 9))
R> msg <- tryCatch(claims_panel(claims, late, t = 3, L = 2, features = "Type"),
+                  error = conditionMessage)
R> writeLines(strwrap(msg, width = 70))

1 payment(s) are dated after t = 3; pass future_payments = "drop" to
discard them

R> claims_panel(claims, late, t = 3, L = 2, features = "Type",
+               future_payments = "drop")$diagnostics$future_amount_dropped

[1] 9
```

4.2. The shipped synthetic portfolio

The package ships a synthetic portfolio from the simulator of Wang and Wüthrich (2022), built on **SynthETIC** and run on a yearly grid with seed 2026: six claim types, twelve accident years, valuation at $t = 12$, horizon $L = 11$. It is a fresh portfolio from the simulator used by the source article, not the source article's own, so the numbers below do not attempt to reproduce that article's. `synthetic_claims` and `synthetic_payments` hold what the insurer knows at the valuation date; `synthetic_claims_full` and `synthetic_payments_full` add the generated ultimate amounts and the payments after the valuation date, for back-testing. The trees see the type, the age, the weekday of the accident and the reporting delay, together with the payment history; the ultimate claim size, which drives the payments, is withheld. The generator violates the model of Section 2 in several ways (inflation, a drifting settlement delay, a claim-size cycle in the accident date, and late-reported claims that are smaller on average), and on the yearly grid its calendar conversion still treats a time unit as a month, which leaves artefacts in the weekday feature and the reporting delay; the back-tests below therefore measure the method under misspecification (Section S1 describes the generator and these mechanisms).

```
R> data("synthetic_claims", "synthetic_payments", package = "baggingReserve")
R> panel <- claims_panel(synthetic_claims, synthetic_payments, t = 12, L = 11,
+                        features = c("Type", "Age", "AccWeekday"))
R> panel
```

```
Claims panel: 71,117 claims reported by t = 12
  accident years 1-12 (I = 12), development years 0-11
  features: Type, Age, AccWeekday (+ reporting delay)
  observed cells: 491,294 of 853,404; recoveries: 528; claims without payment: 8600
```

In the triangle of reporting counts, `claim_counts(panel)`, the reporting delay runs from 0 to 8 years: of the 9 delay columns in that range, 8 contain claims (the column for delay 7 is empty); 8,600 reported claims have no payment yet and 528 of the transactions are recoveries. Fitting one pruned tree per development year takes less than a minute:

```
R> tree <- reserve_tree(panel, seed = 1)
R> tree

Single-tree reserving fit: 11 trees, valuation year t = 12
  (period = development-year index)
  claims: 71,117; features: Type, Age, AccWeekday, rep_delay
  control: cp = 0, xval = 10, minsplit = 20; pruning: min_xerror
 period n_train       cp leaves
      0   41042 0.017310      2
      1   58282 0.008190      5
      2   55733 0.001911     35
      3   50765 0.004043     25
      4   45210 0.015980      9
      5   39296 0.021780      1
      6   33147 0.020130      4
      7   26427 0.012030      1
      8   19887 0.029740      1
      9   13065 0.016880      1
     10    6679 0.093470      1

R> res_tree <- ibnr_reserve(predict(tree), claim_counts(panel))
R> calendar_year_payments(res_tree, "total")[c("13", "total"), ]

       13     total
 51514098 107779181
```

The trees of periods 5, 7, 8, 9 and 10 are pruned to their root nodes: the response is mostly zero and no split survives cross-validation, so these trees predict the training mean for every claim.

A bagged fit with 50 replicates takes a few minutes on two worker processes:

```
R> bag <- bag_reserve(panel, B = 50, seed = 2026, ncores = 2)
R> res50 <- ibnr_reserve(predict(bag))
R> summary(claim_reserves(res50)$reserve)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -8337       0       0    1612     302  127735
```

The last line summarises the reserves of the individual reported claims. The replication script makes the same call with $B = 1000$ replicates and 8 worker processes. Replicate b always uses random-number stream b (Section 3.6), so the first 50 of those replicates are the ones just computed; the next chunk reads that deposit, which is part of the replication material (`results/live/`), not of the package:

```
R> res <- readRDS("results/live/reserve_prediction_live.rds")
R> c(reported = all.equal(res50$reported, res$reported[, 1:50]),
+    ibnr = all.equal(res50$ibnr, res$ibnr[, 1:50]))

reported     ibnr
    TRUE     TRUE

R> smry <- summary(res, probs = c(0.025, 0.5, 0.975))
R> round(smry$tables$total[c("13", "total"), c("mean", "sd", "q2.5%", "q97.5%")])

           mean       sd     q2.5%    q97.5%
13     53163833   933153  51458290  55146743
total 137767028 13006091 117192118 168458852
```

Every training claim receives out-of-bag predictions (the deposit holds the out-of-bag table). The out-of-bag scores are low, between -0.09 and 0.22, and negative in the later development years: the one-year regressions explain little of the variance of next year's payment given the history. The predicted payments of the last calendar years are small and can be negative, because a few recoveries make the training mean of a late tree negative (Section S2).

The uncensored data allow a back-test against the payments that actually followed; Figure 2 shows the predicted payments by calendar year alongside the realised payments, and Table 4 summarises the point predictions:

```
R> data("synthetic_claims_full", "synthetic_payments_full",
+    package = "baggingReserve")
R> bt <- backtest(res, synthetic_claims_full, synthetic_payments_full)
R> d <- as.data.frame(bt)
R> d <- d[d$calendar %in% c("13", "total"), ]
R> transform(d[, c("component", "calendar", "truth", "predicted")],
+    truth = round(truth), predicted = round(predicted),
+    rel_error = round(d$rel_error, 3))

   component calendar     truth predicted rel_error
1   reported       13  51826528  45788120    -0.117
12  reported    total 111241566 115611206     0.039
13      ibnr       13   5318947   7375713     0.387
24      ibnr    total  22742602  22155822    -0.026
25     total       13  57145475  53163833    -0.070
36     total    total 133984168 137767028     0.028
```

The back-test holds one table per component (all of it is printed in Section S2). Next year's total is underpredicted by 7.0%; payments for reported claims are underpredicted by 11.7%

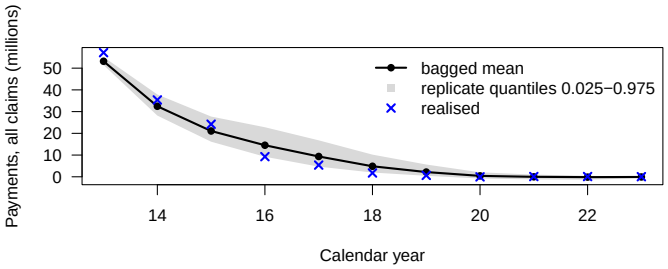


Figure 2: Predicted payments per future calendar year for all claims: the mean over the $B = 1000$ replicates (line and dots), the shaded interval between the 2.5% and 97.5% replicate quantiles (type 7; resampling variation only) and the realised payments (crosses).

	Next year		Through development year 11	
Method	Payments	Rel. error	Payments	Rel. error
Realised	57,145,475	0.0%	133,984,168	0.0%
Single tree	51,514,098	-9.9%	107,779,181	-19.6%
Bagging (mean)	53,163,833	-7.0%	137,767,028	2.8%
Chain ladder	59,679,686	4.4%	140,161,623	4.6%

Table 4: Predicted payments for all claims in the next year (calendar year 13) and in all future years through development year 11, and their relative errors with respect to the realised payments: the single-tree fit, the bagged mean over $B = 1000$ replicates (plug-in completion, the default) and the standard chain ladder on the paid triangle.

and payments for IBNR claims overpredicted by 38.7%, so the component errors partly offset each other. The total through development year 11 is overpredicted by 2.8% (reported-claim payments overpredicted by 3.9%, IBNR payments underpredicted by 2.6%). The realised total lies between the 2.5% and 97.5% replicate quantiles, 117,192,118 and 168,458,852. Next year's realised payments, 57,145,475, exceed all 1000 replicate predictions, which range from 50,602,696 to 56,882,774: the replicate spread measures resampling variation, not forecast error (Section 2.3). The chain ladder's next-year absolute relative error of 4.4% is smaller than the bagged mean's 7.0%; for the total through development year 11 the bagged mean is closer, but by offsetting errors across accident years: it is closer than the chain ladder for 4 of the 11 accident years, and summed in absolute value over the accident years its errors amount to 25,998,854, against 8,588,276 for the chain ladder (Section S2 gives the breakdown); a back-test should look at it as well as at the total. Nor does its small error for the total, against the single-tree fit's 19.6%, show that averaging corrects the single tree: the fits differ in pruning as well, and on every further generator portfolio the bagged mean overpredicts the total by more (Section S4).

The IBNR component is the product of projected counts and a pooled mean payment, so its error has two sources. The chain ladder on the reporting counts projects 2,935 claims to be reported after the valuation date, against 2,588 actually reported later, so the counts are overpredicted by 13.4%; given this count error, the IBNR payments through development year 11, underpredicted by 2.6%, imply that the payment per late-reported claim is underpredicted

by 14.1% ($(1+e_I)/(1+e_N)-1$, with e_I and e_N the relative errors of the payments and of the counts), the error of the pooled mean of Equation (4) on a generator whose late-reported claims are smaller (Section S1).

For totals we recommend simulated completion (Section 3.3); the call differs in one argument, here with the same 50 replicates as above:

```
R> bag_draw <- bag_reserve(panel, B = 50, seed = 2026, ncores = 2,
+    completion = "draw")
R> bt_draw <- backtest(ibnr_reserve(predict(bag_draw)),
+    synthetic_claims_full, synthetic_payments_full)
R> d_draw <- bt_draw$tables$total[c(1, nrow(bt_draw$tables$total)), ]
```

On this misspecified portfolio simulated completion turns the overprediction of the plug-in total into an underprediction (underpredicted by 14.2%, against overpredicted by 2.8% with plugged-in means and $B = 1000$; Section S7); on a correctly specified generator it removes the bias of the plug-in recursion (Section 4.6).

4.3. Back-testing at earlier valuation dates

A user's own data have no realised future, but they can be back-tested by valuing them at an earlier date: the claims of the accident years up to the new valuation date are kept, the constructor keeps only those reported by `t`, `future_payments = "drop"` sets aside the payments after `t`, which then serve as the truth, and the argument `reference` of `backtest()` scores the chain ladder against the same truth. On the shipped portfolio, valued one year earlier:

```
R> cl11 <- synthetic_claims_full[synthetic_claims_full$AccYear <= 11, ]
R> pay11 <- synthetic_payments_full[synthetic_payments_full$Id %in% cl11$Id, ]
R> panel11 <- claims_panel(cl11, pay11, t = 11, L = 10,
+    features = c("Type", "Age", "AccWeekday"), future_payments = "drop")
R> bt11 <- backtest(ibnr_reserve(predict(reserve_tree(panel11, seed = 1))),
+    cl11, pay11, reference = chain_ladder(as_triangle(panel11)))
R> d11 <- as.data.frame(bt11)
R> d11 <- d11[d11$component == "total" & d11$calendar %in% c("12", "total"), ]
R> print(data.frame(calendar = d11$calendar, realised = round(d11$truth),
+    tree = round(d11$predicted), chain_ladder = round(d11$reference)),
+    row.names = FALSE)
```

```
 calendar  realised      tree chain_ladder
       12  50396053  48385097     50311608
    total 116741533 136485448    116133395
```

With $L = 10$, the largest horizon with training claims for its last tree, the script `replication/valuation_t11.R` adds the bagged fit ($B = 1000$): next year's payments are underpredicted by 4.7% by the bagged mean, underpredicted by 4.0% by the single-tree fit and underpredicted by 0.2% by the chain ladder, and the total through development year 10 is overpredicted by 8.5%, overpredicted by 16.9% and underpredicted by 0.5%, respectively. The closest predictor is the chain ladder at both horizons.

Repeated at several valuation dates, the same workflow gives an empirical distribution of the prediction errors, which the replicate quantiles cannot. The script `replication/revision_studies.R` back-tests the single-tree fit, a bagged fit with $B = 1000$ replicates and the chain ladder at $t = 7, \ldots, 11$, each with horizon $L = t - 1$ (Section S2 tabulates the errors). Over the 5 valuation dates the mean absolute relative error of the total is 24.2% for the bagged mean, 22.5% for the single-tree fit and 10.3% for the chain ladder (Table 5); the bagged mean overpredicts the total at every date. The dates share claims and payments, and the horizon changes with t, so these errors are dependent and descriptive. The 2.5% to 97.5% replicate range of the bagged fit contains the realised payments at 2 of the 5 dates next year and 2 for the total. Simulated completion on the same trees removes most of the systematic overprediction (mean signed error of the total 3.2%, against 24.2%), but the errors of single dates remain large and of either sign.

4.4. An illustrative motor third-party liability dataset

A dataset from insurance practice shows how the software handles transaction-level data: the motor third-party liability (MTPL) claims published in a public code repository (Janoušek and Pešta 2026), an illustrative subset of an insurer's data drawn in an undocumented way (shipped with the replication material under its MIT licence), with one row per payment transaction (claim identifier, claim type, accident, reporting and payment dates, amount). We take accident years 2000 to 2009, value the claims at the end of 2009 with $L = 9$ and use the claim type and the accident month as features; the extract ends in 2019, so every predicted cell has its realised value. The script `replication/mtpl_case.R` runs the workflow of Section 4.2 on it (Section S8 shows the calls).

The extract has no claims register, so the claims table is reconstructed from the transactions through 2019: 1,936 of the 29,475 claims reported by 2009 that have no transaction by then are in the data only because they transact later, and the claim type may have been assigned after t. The trees and the IBNR counts are thus conditioned on information from after t (Section S8), and the results illustrate the software, not the accuracy of the method. Its results are in Table 5 and Section S8. The IBNR component underpredicts by 56.0% through development year 9: a claim reported after the valuation date goes on to pay 85,871 on average, against 6,252 for a claim reported by then, a difference the pooled mean payment of Equation (4) cannot reflect. At the 4 valuation dates 2006 to 2009 (Section S8, Table 5) the plug-in bagged mean underpredicts the total at every date and simulated completion raises it at every date: the plug-in recursion errs downwards here, upwards on the generator.

4.5. What the back-tests show

Table 5 summarises the back-tests of the article and the supplementary material that compare all three methods. Over its 4 distinct studies (shipped, $t = 7$–11; generator; changing mix; motor, 4 dates) and the two horizons, the chain ladder has the smallest mean absolute relative error of the three methods in 7 of the 8 comparisons, and in 5 of the 6 without the motor dates; on the generator portfolios it is closer than the bagged mean on 8 of the 9 for next year and on 7 for the total. On all 9 generator portfolios the bagged mean underpredicts next year's payments and overpredicts the total. These findings concern these portfolios and the default configuration, which is not tuned; the single tree is pruned and the bootstrap trees are not, so their difference is not an effect of averaging alone, and a regularised bagged

		Next year				Through development year L				CL closer
Back-test	n	Tree	Bagged	Sim.	CL	Tree	Bagged	Sim.	CL	next, total
Shipped, $t = 12$	1	9.9%	7.0%	7.0%	4.4%	19.6%	2.8%	14.4%	4.6%	1, 0
Shipped, $t = 7$–11	5	8.0%	5.7%	5.7%	5.4%	22.5%	24.2%	8.5%	10.3%	2, 5
Generator	9	11.5%	9.4%	–	5.3%	14.8%	13.5%	–	8.9%	8, 7
Changing mix	5	14.7%	10.7%	–	4.8%	23.3%	4.5%	–	5.1%	5, 3
Motor, $t = 2009$[†]	1	1.3%	0.8%	0.7%	5.1%	2.6%	10.0%	2.6%	2.0%	0, 1
Motor, 4 dates[†]	4	7.8%	7.3%	7.2%	6.0%	6.3%	14.5%	7.9%	4.6%	3, 4

Table 5: Summary of the back-tests: mean absolute relative error of the single-tree fit, the bagged mean with the default plug-in completion (Bagged) and with simulated completion (Sim., where deposited; for the shipped portfolio at $t = 12$ with $B = 100$) and the chain ladder (CL) for all claims, over the n portfolios or valuation dates of each study, and the number of them on which the chain ladder is closer than the plug-in bagged mean (CL closer). Generator: the shipped portfolio and 8 further ones (Section S4); changing mix: Section S6; shipped, $t = 7$ to 11: Section 4.3 and Section S2; motor: Section 4.4 and Section S8. [†]Not clean back-tests: the motor claims table is reconstructed from transactions after the valuation date. The shipped portfolio at $t = 12$ is one of the generator portfolios and the motor case at $t = 2009$ one of the motor dates, and the valuation dates of one portfolio share claims and payments, so the rows are not independent; the means are descriptive. They are computed from unrounded values.

configuration was not examined on them. Part of the bias comes from the plug-in recursion, which simulated completion removes (Sections 4.3 and 4.6).

On every one of the 8 further portfolios, the bagged mean overpredicts next year's IBNR payments, by 23.5% to 46.5% (38.7% on the shipped portfolio). (The late-reported claims of this generator are smaller; Section 2.4.) The realised next-year payments lie within the 2.5% to 97.5% replicate range on 0 of the 8 further portfolios, the realised total on 8. Neither is a coverage rate (Section 2.3): the ranges, on average 6% and 41% of the bagged mean, lie around a biased mean.

A good aggregate can hide a poor allocation: on the shipped portfolio, 33.1% of the predicted reported-claim payments through development year 11 fall on claims already closed at the valuation date, and the open claims are underpredicted by 29.9% (Section S5). On these open claims, the target of **hirem**, its mean absolute relative error over the 9 generator portfolios is 5.6%, against 46.4% for the single-tree fit (Section S7).

4.6. A correctly specified generator

Every generator above violates the model of Section 2, so the back-tests cannot tell errors of the implementation from errors of the method. The script `replication/well_specified.R` simulates portfolios for which the method is correctly specified and the true conditional reserve is known exactly: each of $I = 10$ accident years has n claims of three types, a reporting delay of zero to two years and an age feature without effect. After reporting, a claim pays in development year $l+1$ with probability $a_s(k)\,\delta^l$, where k is its type and $s = \mathbf{1}\{X_l > 0\}$ records whether it paid in year l, and a payment is Gamma distributed with a mean that depends on the type; nothing is paid after $L = 6$. The one-year conditional mean is thus piecewise

		Total reserve			$n = 50{,}000$
Persistence	Estimator	$n = 2{,}000$	$n = 10{,}000$	$n = 50{,}000$	Next year
strong	Single tree, plug-in	41.0 (1.6)	60.3 (0.7)	62.1 (0.1)	0.0 (0.0)
	Single tree, simulated	0.3 (0.3)	0.8 (0.1)	0.8 (0.1)	0.0 (0.0)
	Bagged mean, plug-in	40.5 (0.6)	50.0 (0.4)	53.7 (0.2)	0.9 (0.1)
	Bagged mean, simulated	1.9 (0.3)	2.2 (0.2)	1.7 (0.1)	0.9 (0.1)
	True means, plug-in	61.8 (0.2)	62.0 (0.1)	62.0 (0.0)	0.0
moderate	Single tree, plug-in	4.2 (0.6)	11.0 (0.4)	16.5 (0.1)	0.0 (0.0)
	Single tree, simulated	0.2 (0.3)	0.0 (0.1)	0.0 (0.1)	0.0 (0.0)
	Bagged mean, plug-in	13.5 (0.5)	15.7 (0.2)	17.4 (0.2)	1.0 (0.0)
	Bagged mean, simulated	2.3 (0.3)	1.9 (0.1)	1.5 (0.1)	1.0 (0.0)
	True means, plug-in	17.9 (0.0)	17.9 (0.0)	17.9 (0.0)	0.0
none	Single tree, plug-in	0.0 (0.2)	−0.1 (0.1)	0.0 (0.1)	0.0 (0.0)
	Single tree, simulated	0.1 (0.2)	−0.1 (0.1)	0.0 (0.1)	0.0 (0.0)
	Bagged mean, plug-in	3.9 (0.2)	3.9 (0.1)	2.9 (0.1)	0.7 (0.0)
	Bagged mean, simulated	2.4 (0.2)	1.6 (0.1)	1.0 (0.1)	0.7 (0.1)
	True means, plug-in	0.0	0.0	0.0	0.0

Table 6: Correctly specified generator: relative bias (%) of the predicted payments of the reported claims against the true conditional reserve, mean over 20 portfolios per size n (claims per accident year; Monte Carlo standard error in parentheses), for the total through development year 6 and, at the largest size, for next year's payments. Plug-in: completion by the predicted means; simulated: `completion = "draw"`; true means, plug-in: the plug-in recursion applied to the true one-year means, without estimation. $B = 100$ replicates, $R = 50$ simulated paths for the single tree.

constant in the type and in $\mathbf{1}\{X_l > 0\}$, the same in every accident year, and a regression tree represents it exactly. Given the information at $t = 10$, each reported claim's expected future payments follow from an exact recursion over the two payment states (verified by simulation and by enumerating all payment paths). Three scenarios differ only in how much a payment in one year raises the chance of a payment in the next: strongly, moderately, or not at all ($a_1 = a_0$). The estimators are scored on the reported claims against this truth (IBNR claims, for which the pooling condition of Section 2.4 fails by construction, are left out).

Next year's payments use only observed histories, and the single-tree fit recovers them: at $n = 50{,}000$ its bias is at most 0.05% in absolute value in every scenario, with Monte Carlo standard errors of at most 0.05%. The bagged mean overpredicts them slightly, by 0.7% to 1.0% with Monte Carlo standard errors of at most 0.1%, also where the history carries no information, so this small upward bias of the averaged unpruned trees remains at the largest portfolio; the pruned single tree, fitted without the bootstrap, shows no such bias, and where the history carries no information it is also unbiased for the total (−0.1% to 0.0% across the sizes). An ablation on 20 portfolios per scenario with $n = 10{,}000$ (`replication/well_specified_ablation.R`, Section S9) varies the bootstrap trees: their next-year bias without and with strong persistence is +1.21% and +1.27% with **rpart**'s default minimum leaf of 7 claims, +0.27% and +0.41% with 25, −0.01% and +0.15% with 100, and +1.13% and +1.17% when every bootstrap tree is pruned by cross-validation, against −0.03% and +0.11% for the pruned single tree. The bias lies in the small leaves of the unpruned bootstrap trees: with a minimum leaf of 100 claims it vanishes within Monte Carlo error, whereas pruning

each bootstrap tree at the smallest cross-validated error of its own sample hardly changes it, presumably because copies of a claim drawn into both the training and the validation folds make the cross-validation favour large trees. Without persistence the total is affected alike: +3.90% with the default leaves, +0.30% with a minimum leaf of 100. A user of the bagged mean can therefore set a larger `minbucket` in `control`. Where the history carries information, the plug-in recursion is biased for the total, and the bias grows with the portfolio instead of vanishing: at $n = 50{,}000$ the bias of the single-tree fit is +62.1% with strong and +16.5% with moderate persistence, against +62.0% and +17.9% for the plug-in recursion on the true means, which it approaches with strong persistence; the bagged mean behaves alike (+53.7% and +17.4%). With simulated completion, the bias of the total is at most 1.7% in absolute value for both estimators in every scenario at $n = 50{,}000$. The plug-in recursion replaces a claim's uncertain next payment by its mean, a small positive amount that the following trees treat like an observed payment, so that with positive persistence a plugged-in mean crosses the split between paying and non-paying claims and overstates later payments; the same mechanism contributes to the upward bias of the bagged totals in the back-tests above, and a smoother dependence on the history, not examined here, may give a smaller bias. The plug-in default is therefore not a consistent estimator of the multi-year reserve where the occurrence of a payment depends sharply on the history, and for totals we recommend `completion = "draw"`, although on the shipped portfolio and the motor extract it does not make the bagged mean more accurate than the chain ladder (Section S7, Section S8).

4.7. Replicating this paper

Every number, table and figure of the article and of its supplementary material is read from the deposits in `results/` of the replication material or computed while knitting; knitting stops if a file of a deposit differs from its checksum manifest, and an absent deposit is marked in bold. The `Makefile` in `paper/` has three levels: `make render` knits both documents from the deposits, computing only the small examples; `make replicate` first recomputes the deposits of the main examples (about half an hour on four cores); `make replicate-all` also recomputes every study summarised in Section 4.5, which takes hours. The correctly specified simulation of Section 4.6 is not part of it: its full grid takes about 900 core-hours and was run as a Slurm job array (`replication/well_specified.slurm`), as is its leaf-size ablation (`replication/well_specified_ablation.R`); `WS_QUICK=1 make well-specified` and `WSA_QUICK=1 make well-specified-ablation` run small versions in minutes. The standalone script `code.R` runs the displayed code and recomputes the main examples in a scratch copy, comparing them with the deposits to a relative tolerance of 10^{-10}. With `CODE_QUICK=1` it takes minutes and compares only the first $B = 50$ replicates of the live run and of the motor case with the deposited ones; its recomputation at $t = 11$ is printed, not compared. The package's vignette `baggingReserve-jss`, this article as a precompiled PDF, is not re-executed by `R CMD check`.

5. Summary and discussion

baggingReserve provides a data contract, a recursion engine with a learner interface and two completion modes, and back-tests, with the source article's method as the documented default baseline. On the portfolios studied here the standard chain ladder is the more accurate method on most of them (Section 4.5); the bagged trees are biased with a consistent sign

on the generator, partly through the plug-in recursion, and good aggregates can rest on offsetting errors, across accident years, between components and between open and closed claims (33.1% of the reported-claim reserve of the shipped portfolio falls on claims already closed). On a correctly specified generator the plug-in completion is biased for the total where the occurrence of a payment depends on the payment history, and simulated completion, which we recommend for totals, removes most of the bias.

The limitations are those of the design. Only static features enter the panel, so neither a settlement status that changes over time nor case reserves can be used, and the package cannot allocate the reserve between open and closed claims; a settlement layer as in **hirem** would be needed. The IBNR component holds the projected counts fixed and pools the payments of the reported claims, which fails where late-reported claims differ in kind; an external count projection, from **ReSurv** say, has to be combined with it outside the package. The trees minimise squared error, which gives large payments a strong influence. The package delivers no predictive distribution or risk measure of the reserve (Section 2.3). The evaluation is limited as well: the claim-level estimators are compared with the plain chain ladder only, not with a stronger aggregate method such as the double chain ladder or a bootstrap chain ladder, **hirem** only on the open claims, and a tuned configuration of the trees was not examined.

A claim-level method can be expected to do better where the business mix changes, the features carry real signal, reported claims carry most of the liability and the late development years have enough paying claims; these are expectations, not results of this paper (the one test of a changing mix, Section S6, was mixed), and `backtest()` at several valuation dates checks them.

Computational details

The results in this paper were obtained using R 4.6.1 (R Core Team 2026) with **baggingReserve** 0.2.0 and **rpart** 4.1.27. The package datasets were generated once with **SynthETIC** 1.1.2 (Avanzi *et al.* 2021; Avanzi, Ho, Taylor, Wang, and Wong 2026) and the bundled copy of the generator of Wang and Wüthrich (2022), with its commit identifier recorded in `data-raw/GENERATION.txt` of the replication material. The paper is knitted with **knitr** (Xie 2026, 2015), parallel execution uses the **parallel** package of R, and the package tests use **testthat** (Wickham 2011). All deposited results were computed with **baggingReserve** 0.2.0 on Linux (Rocky Linux 9.8, AMD EPYC and Intel Xeon nodes of the MFF Chimera cluster). To check portability, the live run, the second portfolio and the valuation at $t = 11$ were rerun on macOS 26.6 (Apple M5); the 2,194,562 compared values agree to a relative difference of at most 1.5×10^{-12}, the level of floating-point rounding. Earlier deposits computed on macOS agree with these to a relative difference of order 10^{-12} (`results/reproduction/`).

R itself and the packages on which **baggingReserve** depends are available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/`; **baggingReserve** is available from CRAN at `https://CRAN.R-project.org/package=baggingReserve`; its development version is at `https://github.com/HeirOfIsildur/baggingReserve`.

Acknowledgements

The research of Jan Janoušek and Michal Pešta was supported by the Czech Science Foun-

dation project GAČR No. 27-25747L and by the Charles University Grant Agency project GA UK No. 188325.

Declaration of generative AI and AI-assisted technologies

During the preparation of this work the authors used generative AI tools to improve their English as non-native speakers, to check derivations, formulas and arithmetic, and to assist with writing and reviewing the analysis code; the exact reserves of Section 4.6 were also verified symbolically (`replication/well_specified_truth.wl`). The authors reviewed and edited the output and take full responsibility for the content of the publication.

References

Avanzi B, Ho W, Taylor G, Wang M, Wong B (2026). ***SynthETIC**: Synthetic Experience Tracking Insurance Claims*. doi:10.32614/CRAN.package.SynthETIC. R package version 1.1.2, URL https://CRAN.R-project.org/package=SynthETIC.

Avanzi B, Taylor G, Wang M (2023). "**SPLICE**: A Synthetic Paid Loss and Incurred Cost Experience Simulator." *Annals of Actuarial Science*, **17**(1), 7–35. doi:10.1017/S1748499522000057.

Avanzi B, Taylor G, Wang M, Wong B (2021). "**SynthETIC**: An Individual Insurance Claim Simulator with Feature Control." *Insurance: Mathematics and Economics*, **100**, 296–308. doi:10.1016/j.insmatheco.2021.06.004.

Baudry M, Robert CY (2019). "A Machine Learning Approach for Individual Claims Reserving in Insurance." *Applied Stochastic Models in Business and Industry*, **35**(5), 1127–1155. doi:10.1002/asmb.2455.

Breiman L (1996). "Bagging Predictors." *Machine Learning*, **24**(2), 123–140. doi:10.1007/BF00058655.

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984). *Classification and Regression Trees*. Wadsworth, Belmont.

chainladder Developers (2026). ***chainladder**: Actuarial Reserving in Python*. Python package version 0.10.1, URL https://github.com/casact/chainladder-python.

Crevecoeur J, Robben J (2022). ***hirem**: Hierarchical Reserving Models*. R package version 0.1.0, URL https://github.com/jonascrevecoeur/hirem.

Crevecoeur J, Robben J, Antonio K (2022). "A Hierarchical Reserving Model for Reported Non-Life Insurance Claims." *Insurance: Mathematics and Economics*, **104**, 158–184. doi:10.1016/j.insmatheco.2022.02.005.

Delong Ł, Lindholm M, Wüthrich MV (2022). "Collective Reserving Using Individual Claims Data." *Scandinavian Actuarial Journal*, **2022**(1), 1–28. doi:10.1080/03461238.2021.1921836.

Duval F, Pigeon M (2019). "Individual Loss Reserving Using a Gradient Boosting-Based Approach." *Risks*, **7**(3), 79. doi:10.3390/risks7030079.

England PD, Verrall RJ (2002). "Stochastic Claims Reserving in General Insurance." *British Actuarial Journal*, **8**(3), 443–518. doi:10.1017/S1357321700003809.

Gabrielli A, Richman R, Wüthrich MV (2020). "Neural Network Embedding of the Over-Dispersed Poisson Reserving Model." *Scandinavian Actuarial Journal*, **2020**(1), 1–29. doi:10.1080/03461238.2019.1633394.

Gesmann M, Murphy D, Zhang Y, Carrato A, Wüthrich M, Concina F, Dal Moro E (2026). ***ChainLadder**: Statistical Methods and Models for Claims Reserving in General Insurance*. doi:10.32614/CRAN.package.ChainLadder. R package version 0.2.21, URL https://CRAN.R-project.org/package=ChainLadder.

Hiabu M, Hofman E, Pittarello G (2023). "A Machine Learning Approach Based on Survival Analysis for IBNR Frequencies in Non-Life Reserving." arXiv:2312.14549. doi:10.48550/arXiv.2312.14549.

Hofman E, Pittarello G, Hiabu M (2026). ***ReSurv**: Machine Learning Models for Predicting Claim Counts*. doi:10.32614/CRAN.package.ReSurv. R package version 1.1.0, URL https://CRAN.R-project.org/package=ReSurv.

Janoušek J, Pešta M (2025). "Bagging and Regression Trees in Individual Claims Reserving." *Statistical Papers*, **66**(4), 89. doi:10.1007/s00362-025-01715-9.

Janoušek J, Pešta M (2026). `rf-pipeline-calibration-reserving`*: Code and Illustrative Motor Third-Party Liability Data*. Version 1.0, commit d259117, URL https://github.com/HeirOfIsildur/rf-pipeline-calibration-reserving.

L'Ecuyer P, Simard R, Chen EJ, Kelton WD (2002). "An Object-Oriented Random-Number Package with Many Long Streams and Substreams." *Operations Research*, **50**(6), 1073–1075. doi:10.1287/opre.50.6.1073.358.

Liaw A, Wiener M (2002). "Classification and Regression by **randomForest**." *R News*, **2**(3), 18–22. URL https://CRAN.R-project.org/doc/Rnews/.

Lopez O, Milhaud X, Thérond PE (2016). "Tree-Based Censored Regression with Applications in Insurance." *Electronic Journal of Statistics*, **10**(2), 2685–2716. doi:10.1214/16-EJS1189.

Lopez O, Milhaud X, Thérond PE (2019). "A Tree-Based Algorithm Adapted to Microlevel Reserving and Long Development Claims." *ASTIN Bulletin*, **49**(3), 741–762. doi:10.1017/asb.2019.12.

Mack T (1993). "Distribution-Free Calculation of the Standard Error of Chain Ladder Reserve Estimates." *ASTIN Bulletin*, **23**(2), 213–225. doi:10.2143/AST.23.2.2005092.

Martínez Miranda MD, Nielsen JP, Verrall R (2012). "Double Chain Ladder." *ASTIN Bulletin*, **42**(1), 59–76. doi:10.2143/AST.42.1.2160712.

Martínez-Miranda MD, Nielsen JP, Verrall R (2022). ***DCL**: Claims Reserving under the Double Chain Ladder Model*. doi:10.32614/CRAN.package.DCL. R package version 0.1.2, URL https://CRAN.R-project.org/package=DCL.

Peters A, Hothorn T (2026). ***ipred**: Improved Predictors*. doi:10.32614/CRAN.package.ipred. R package version 0.9-16, URL https://CRAN.R-project.org/package=ipred.

R Core Team (2026). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Therneau T, Atkinson B (2026). ***rpart**: Recursive Partitioning and Regression Trees*. doi:10.32614/CRAN.package.rpart. R package version 4.1.27, URL https://CRAN.R-project.org/package=rpart.

Verrall RJ, Wüthrich MV (2016). "Understanding Reporting Delay in General Insurance." *Risks*, **4**(3), 25. doi:10.3390/risks4030025.

Wang M, Wüthrich MV (2022). "Individual Claims Generator for Claims Reserving Studies: Data Simulation.R." *Technical Report 4127073*, SSRN. doi:10.2139/ssrn.4127073.

Wickham H (2011). "**testthat**: Get Started with Testing." *The R Journal*, **3**(1), 5–10. doi:10.32614/RJ-2011-002.

Wright MN, Ziegler A (2017). "**ranger**: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R." *Journal of Statistical Software*, **77**(1), 1–17. doi:10.18637/jss.v077.i01.

Wüthrich MV (2018). "Machine Learning in Individual Claims Reserving." *Scandinavian Actuarial Journal*, **2018**(6), 465–480. doi:10.1080/03461238.2018.1428681.

Xie Y (2015). *Dynamic Documents with R and **knitr***. 2nd edition. Chapman and Hall/CRC, Boca Raton.

Xie Y (2026). ***knitr**: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.52, URL https://yihui.org/knitr/.

Affiliation:

Jan Janoušek, Michal Pešta
Department of Probability and Mathematical Statistics
Faculty of Mathematics and Physics
Charles University
Sokolovská 49/83, 186 00 Prague, Czech Republic
E-mail: janousek@karlin.mff.cuni.cz, pesta@karlin.mff.cuni.cz